Goddard
270946
P. 227

SOFTWARE ENGINEERING LABORATORY

SEL-88-005

PROCEEDINGS OF THE FIRST NASA ADA USERS' SYMPOSIUM

DECEMBER 1988

(NASA-TM-102941) PROCEEDINGS OF THE FIRST NASA Ada USERS' SYMPOSIUM (NASA) 227 p CSCL 09B

N91-11389

Unclas
G3/61 0270946

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

PROCEEDINGS
OF THE
FIRST NASA ADA USERS' SYMPOSIUM

Organized by:
Software Engineering Laboratory
GSFC

Sponsored by:
Goddard Ada Users' Group

December 1, 1988

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development Branch)
The University of Maryland (Computer Sciences Department)
The Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models in the process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

Single copies of this document can be obtained from:

NASA/Goddard Space Flight Center
Systems Development Branch
Code 552
Greenbelt, Maryland 20771

TABLE OF CONTENTS
OF THE
FIRST NASA ADA USERS' SYMPOSIUM

TABLE OF CONTENTS

INTRODUCTION
OF
FIRST NASA ADA USERS' SYMPOSIUM

by

E. Seidewitz
NASA/Goddard Space Flight Center

INTRODUCTION

The Ada programming language was created as the common language for the Department of Defense (DOD). However, there are a growing number of organizations outside the DOD, both government and commercial, who are choosing to use Ada for their large system development efforts. NASA is one such organization. Mandated for the space station, Ada has also been adopted or considered for use by several other large NASA programs.

Ada has the potential to be a part of the most significant change in software engineering technology within NASA in the last twenty years. Thus, it is particularly important that all NASA centers be aware of Ada experience and plans at other centers. To promote such an awareness, the First NASA Ada Users' Symposium provided a forum for the exchange of ideas, experiences and plans on the use of Ada within NASA.

The symposium attracted a diverse, enthusiastic audience. The program covered Ada activity across NASA, with presenters representing five of the nine major NASA centers and the Space Station Freedom Program Office. Projects discussed included:

- Space Station Freedom Program Office: the implications of Ada on training, reuse, management and the software support environment;
- Johnson Space Center (JSC): early experience with the use of Ada, software engineering and Ada training and the evaluation of Ada compilers;
- Marshall Space Flight Center (MSFC): university research with Ada and the application of Ada to Space Station Freedom, the Orbital Maneuvering Vehicle, the Aero-Assist Flight Experiment and the Secure Shuttle Data System;
- Lewis Research Center (LeRC): the evolution of Ada software to support the Space Station Power Management and Distribution System;
- Jet Propulsion Laboratory (JPL): the creation of a centralized Ada development laboratory and current applications of Ada including the Real-time Weather Processor for the FAA;
- Goddard Space Flight Center (GSFC): experiences with Ada in the Flight Dynamics Division and the Extreme Ultraviolet Explorer (EUVE) project and the implications of GSFC experience for Ada use in NASA.

Despite the diversity of the presentations, several common themes emerged from the program:

- Methodology: NASA experience in general indicates that the effective use of Ada requires modern software engineering methodologies. There is a growing trend towards the acceptance of object-oriented approaches as the basis for the most appropriate methodologies for Ada development.
- Training: It is the software engineering principles and methods that surround Ada, rather than Ada itself, which requires the major training effort. This is evident in experience at LeRC, JPL and GSFC and is reinforced by the research of the University of Houston for JSC. Further, both GSFC and the University of Houston stress that this training must be focused to the needs of each organization and must include immediate hands-on involvement in real development efforts.
- Reuse: Due to training and transition costs, the use of Ada may <u>initially</u> actually decrease productivity, as was clearly found at GSFC. However, at GSFC as well as in work done for JSC, there is a clear indication that the use of Ada and associated methodologies can result in an immediate significant increase in the reusability of software. Of course, over time this will result in a major increase in effective productivity, reliability and maintainability, since less and less new code will need to be created for each project.

- Real-time:—Work, at LeRC, JPL and GSFC shows that it is possible to use Ada for real-time applications. However, the LeRC experience especially shows how careful one must be in choosing a compiler. At GSFC, the EUVE project found it necessary to modify the vendor-supplied run-time system to handle a specific embedded hardware configuration.

Overall, the symposium reflected a high level of enthusiasm for the use of Ada in NASA. Ada is being effectively applied to flight and ground-support tasks, both inside and outside the space station project. However, there are also some cautionary notes: the transition to Ada may take longer and be more difficult than originally anticipated; NASA needs to focus more clearly, effectively and intensely on software engineering training efforts; and NASA must press compiler vendors to provide more high-quality Ada compilers with the features needed for real-time, embedded applications.

By providing a forum for discussing Ada benefits, lessons-learned and problems, the First NASA Ada Users' Symposium was highly successful in its aim of fostering communication between the NASA community of Ada users. This community is still young and growing, but it is clear that Ada is "here to stay" in NASA. Right now we are at the knee of the growth curve in the use of Ada. As we proceed upward on that curve it will be increasingly important to maintain and strengthen the sharing of experience. This symposium will have been truly successful if it is only a beginning to such a process.

In conclusion, I would like to greatly thank Lisa Kelly, Frank McGarry and the Software Engineering Laboratory staff. Without their help it would have been totally impossible to organize this symposium in the short time we did. I would also like to thank all the presenters who, on quite short notice, put together an excellent overview of Ada activities in NASA.

Ed Seidewitz
Head, Goddard Ada Users' Group
Goddard Space Flight Center

Session 1: EXPERIENCES

1. Ed Seidewitz, NASA/GSFC
2. William Howle, NASA/MSFC
3. Robert Loesh and Pat Molko, JPL

NASA

MISSION OPERATIONS
AND DATA SYSTEMS DIRECTORATE

GSFC

WELCOME

to the

First NASA Ada Users' Symposium

sponsored by

THE GODDARD ADA USERS' GROUP

Experiences With Ada in the Flight Dynamics Division

Ed Seidewitz
Code 554

FLIGHT DYNAMICS SOFTWARE CHARACTERISTICS

COMPONENT	TYPE	SIZE (SLOC)	% REUSED EACH MISSION	DEVELOPMENT DURATION	EFFORT (PER MISSION)
ATTITUDE - DETERMINATION - CONTROL - CALIBRATION - SIMULATION ETC.	MISSION UNIQUE	250,000	25%	27 MO.	40 MY
ORBIT / TRACKING DATA PROCESSING	MISSION GENERAL	1,200,000	95%+	12-18 MO.	2 MY
MISSION DESIGN / ANALYSIS	MISSION GENERAL	200,000	85%	12-18 MO.	5 MY
ORBIT MANEUVER SUPPORT	MISSION GENERAL	100,000	60%	12-18 MO.	5 MY

STUDY OF Ada AS A "METHODOLOGY"

PROJECT (GRO DYNAMIC SIMULATOR)

• SIZE	45,000 (FORTRAN) SLOC
• DURATION	24 - 30 MONTHS
• ENVIRONMENT	VAX 11/780-VAX 8600
• STAFFING	7 PEOPLE
• EFFORT	8 - 10 MY

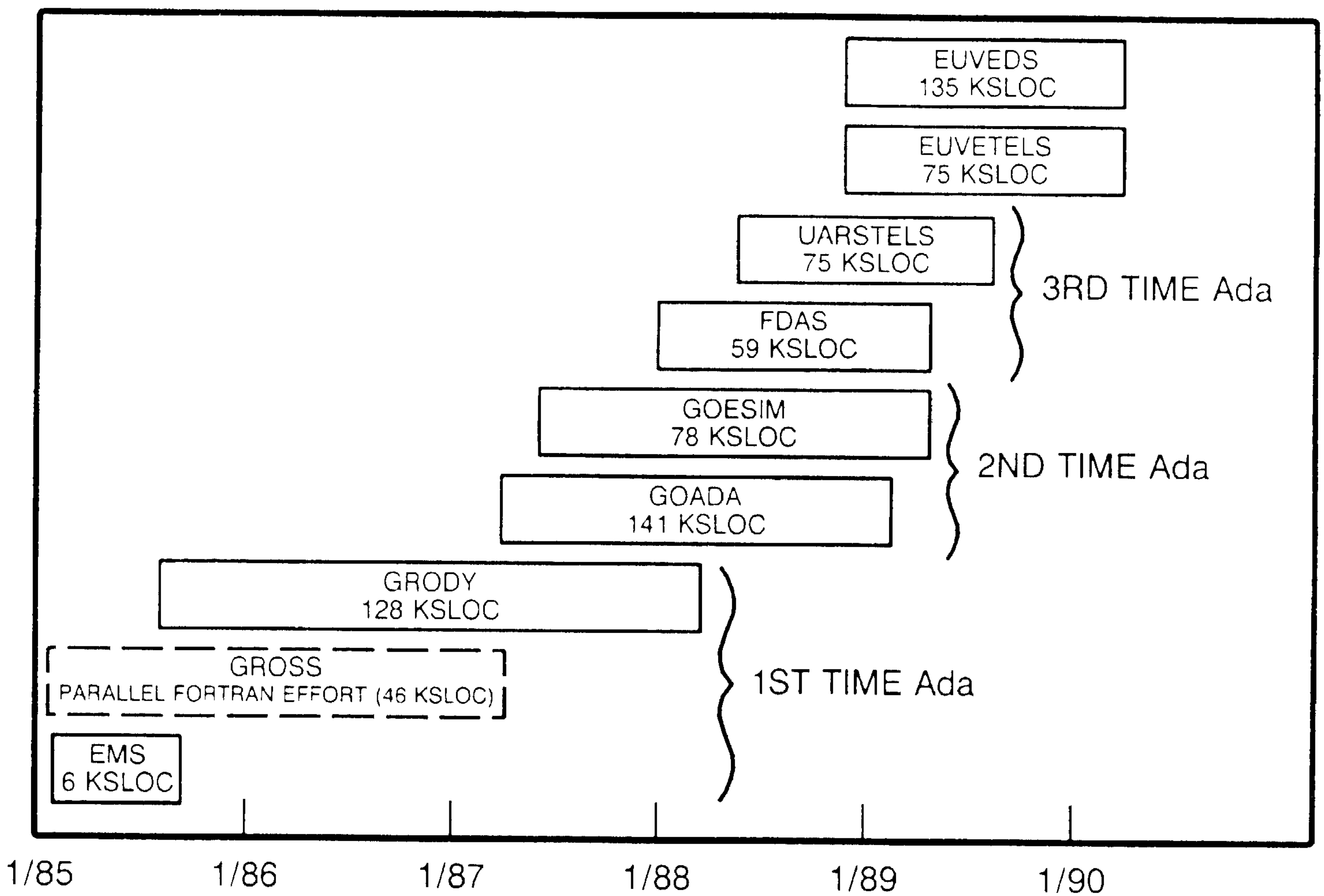
OBJECTIVES

- DETERMINE VALUE OF Ada FOR NASA GROUND SYSTEMS
- ASSESS EFFECTIVENESS OF OOD, PAM, CSM
- DEVELOP APPROACHES FOR REUSABLE SOFTWARE
- DEVELOP MEASURE (CRITERIA) FOR Ada/SPACE STATION

APPROACH

- 2 PARALLEL DEVELOPMENT EFFORTS (FORTRAN AND Ada)
- EXTENSIVE TRAINING FOR Ada TEAM
- CLOSELY MONITOR PROCESS AND PRODUCT
- DEVELOP MEASURES AND COMPARE 2 PRODUCTS

Ada PROJECTS IN FLIGHT DYNAMICS DIVISION



D217.003

Ada PROJECTS STUDIED

PROJECT	TYPE	SIZE* (SLOC)	START DATE	DURATION	(11/30/88) STATUS	STAFF LEVEL
EMS	ELECTRONIC MAIL (PRACTICE/TRAINING)	5730	3/85	4 MO.	COMPLETE	7
GRODY	SIMULATOR (FLIGHT CONTROL SYSTEM)	128000	8/85	36 MO.	COMPLETE	7
GOADA	SIMULATOR (FLIGHT CONTROL SYSTEM)	141000	7/87	20 MO.	SYSTEM TEST	7
GOESIM	SIMULATOR (TELEMETRY)	78000	9/87	18 MO.	SYSTEM TEST	4
FDAS	EXECUTIVE (SOURCE CONTROL)	58700	1/88	13 MO.	SYSTEM TEST	4
UARSTELS	SIMULATOR (TELEMETRY)	75000	2/88	18 MO.	CODE	3

*SLOC = TOTAL LINES (CARRIAGE RETURNS) INCLUDES COMMENTS/BLANKS/REUSED
ALL PROJECTS DEVELOPED ON DEC VAX 11/780 OR VAX 8600

D217.0104

DOCUMENTATION OF ADA EXPERIENCE

- "GENERAL OBJECT-ORIENTED SOFTWARE DEVELOPMENT" METHODOLOGY DESCRIPTION
- "ADA STYLE GUIDE"
- "ADA TRAINING EVALUATION AND RECOMMENDATIONS"
- "ASSESSING THE ADA DESIGN PROCESS AND ITS IMPLICATIONS"
- LESSONS LEARNED DURING CODING AND UNIT TESTING
- LESSONS LEARNED DURING SYSTEM TESTING
- "EVOLUTION OF ADA TECHNOLOGY FOR FLIGHT DYNAMICS"

LESSONS LEARNED DURING ADA TRAINING

- KEY PROBLEM AREAS FOR ADA LANGUAGE TRAINING
  - Input / Output
  - Data Types
  - Generics
  - Tasking
  - Library Structure

- TRAINING MUST BE DRIVEN BY THE ENVIRONMENT
  - GRODY Team Received 6 Mo. Of Intensive Training (ALSYS Videos, Booch, PAMELA, Training Project)
  - Later Teams Received ~1 Mo. Of Focused Training (Syntax Course, Application Examples, Specific Methodology)

- TRAINING MUST BE IN CONJUNCTION WITH OR IMMEDIATELY FOLLOWED BY ACTUAL PROJECT EXPERIENCE

- THE MAJOR DIFFICULTY IN TRAINING IS LEARNING A NEW METHODOLOGY, NOT LEARNING ADA

LESSONS LEARNED DURING ADA DESIGN

- THE SYSTEM SPECIFICATION MAY BE BIASED TOWARDS PREVIOUS DESIGN APPROACHES
- METHODOLOGY IS IMPORTANT
  - THE METHODOLOGY SHOULD BE CHOSEN EARLY
  - BOTH DEVELOPERS AND MANAGERS SHOULD UNDERSTAND THE METHODOLOGY
  - THE METHODOLOGY SHOULD EXPLOIT ADA'S FEATURES
  - THE "GENERAL OBJECT-ORIENTED DESIGN" METHODOLOGY HAS BEEN SUCCESSFUL
- A "COMPILABLE DESIGN" IS VERY USEFUL FOR DESIGN VALIDATION AND DOCUMENTATION
- THERE IS A COST ASSOCIATED WITH DISCARDING PREVIOUS DEVELOPMENT LEGACY
- SOME CHANGE IS NEEDED IN THE TRADITIONAL LIFE CYCLE MODEL

Ada FEATURES

	IMPLEMENTATION EASE	BENEFICIAL
TASKING	-	+
GENERICS	+	+ +
STRONG TYPING	0	0
EXCEPTION HANDLING	0	+
NESTING	+	-
SEPARATE SPECS/BODIES	+ +	+ +

* SUBJECTIVE ASSESSMENTS BASED ON INTERVIEWS

G58.008

Application of Ada to MSFC Projects

First NASA Ada User's Symposium
NASA/GSFC
December 1,1988

William T. Howle
NASA/MSFC

TOPICS

- UNIVERSITY RESEARCH
- SPACE STATION FREEDOM
- ORBITAL MANEUVERING VEHICLE (OMV)
- AERO-ASSIST FLIGHT EXPERIMENT (AFE)
- SECURE SHUTTLE DATA SYSTEM (SSDS)

AUBURN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NASA PROJECTS

QUEST

- Query Utility Environment for Software Testing
  Dr. David B. Brown, Principal Investigator

GRASP

- Graphical Representation of Algorithms, Structures and Processes
  Dr. James C. Cross, Principal Investigator

Query
Utility
Environment for
Software
Testing

GENERAL GOAL:

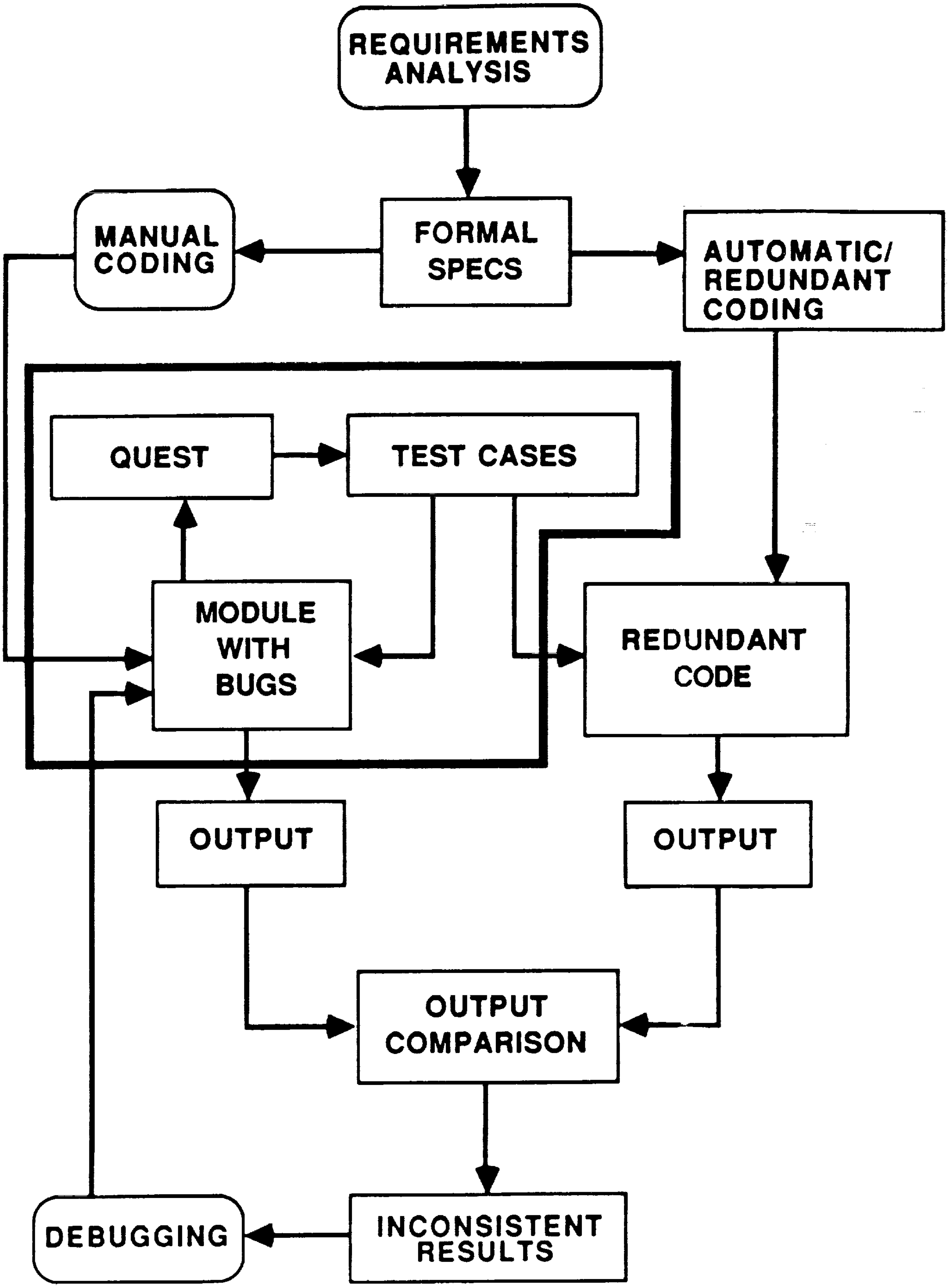
To provide an environment in which more tests and more effective tests can be performed in order to increase the reliability of Ada code.

Query
Utility
Environment for
Software
Testing

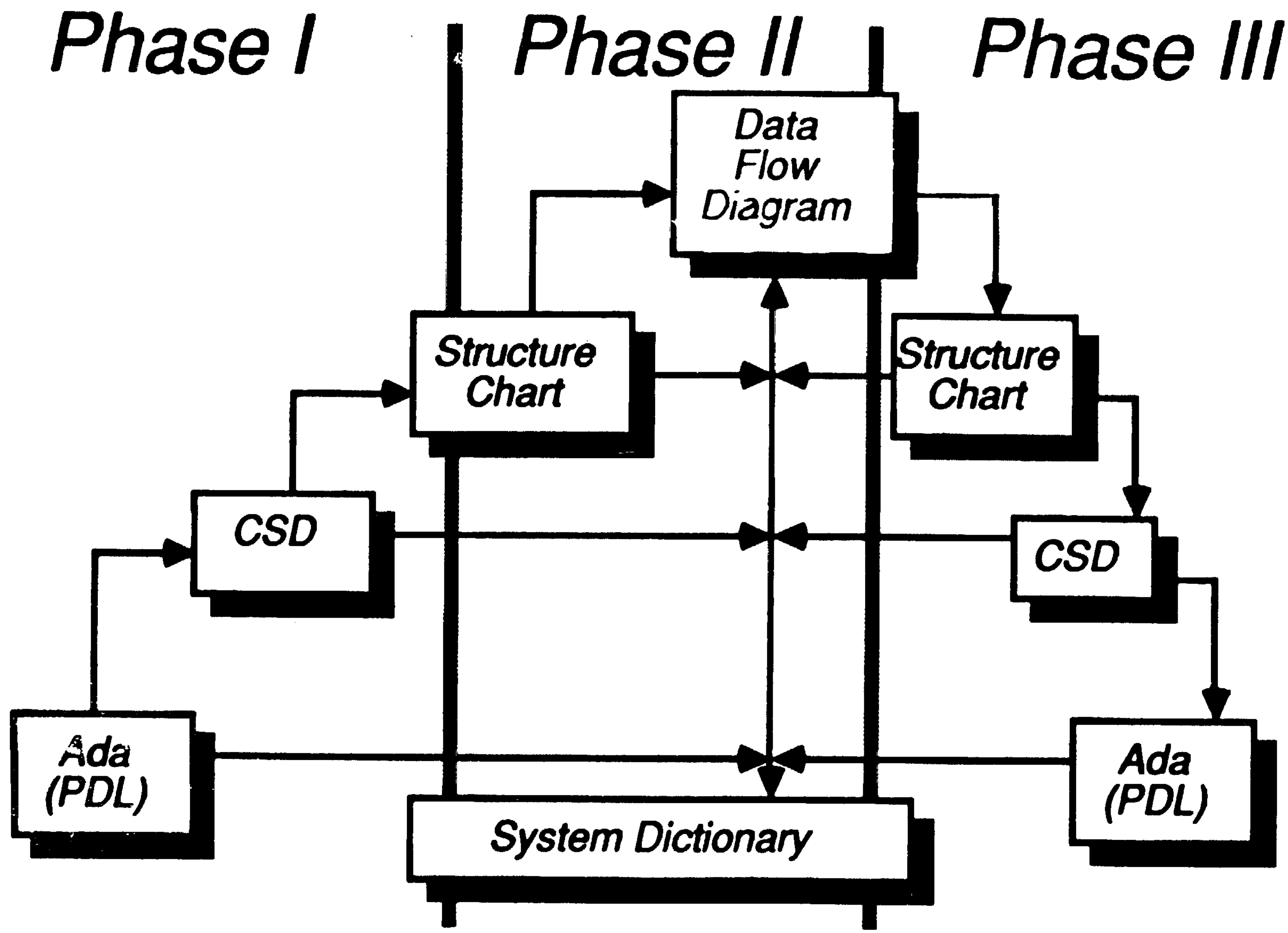
OBJECTIVES:

1. Intelligent Automatic Test Case Generation
2. Controlled Test Case Execution
3. Coverage Analysis
   - to measure module reliability

QUEST/ADA PROJECT SCOPE



GRASP/Ada



Phase I
Phase II
Phase III
Data Flow Diagram
Structure Chart
Structure Chart
CSD
CSD
Ada (PDL)
Ada (PDL)
System Dictionary

GRASP OVERVIEW

GENERAL GOAL:

To increase designer and programmer productivity through the use of graphics-based tools.

OBJECTIVES:

To produce immediate graphical aids for development and maintenance.

. CSD - Control Structure Diagram

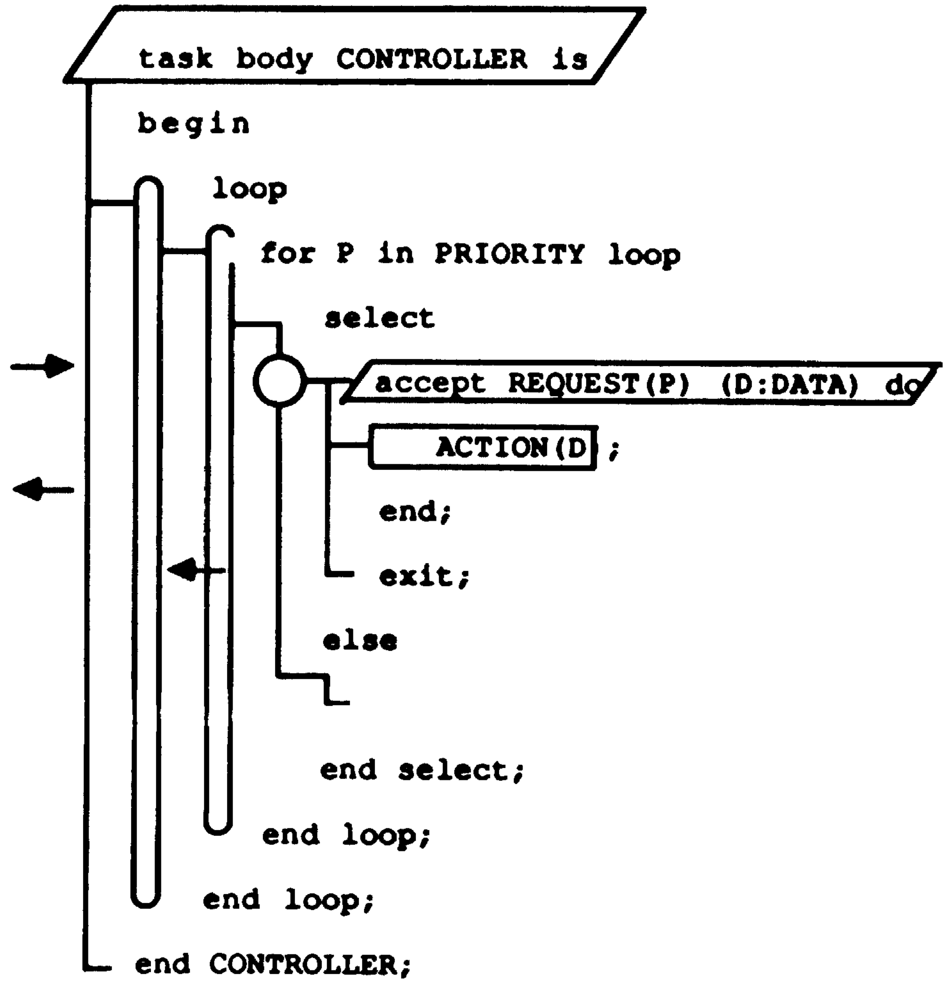
. Structure Chart

. Data Flow Diagram

To understand the process of graphical representation generation.

To reverse the process to generate code from the graphical representation.

An Example (from J. G. P. Barnes)



Justification

- Provides an understanding of automatic code generation
- Ada is well-defined – a good base from which to work upwards
- Many designers will work in an Ada PDL
- GR's can be provided at little cost
- Reviews of requirements and design specifications are potentially better facilitated by use of GR's
- 90% of maintenance effort is attempting to understand existing code (Standish)
- Generation of standardized GR's of Ada software will promote reuseability – an original objective for the adoption of Ada

- Software safety can be ensured to a large extent by software verification

"A Comparison of Software Verification Technique" (NASA Goddard SE Lab series, April 1985)

- Empirical study of code-reading, functional testing, and structural testing

- Found code-reading provided greatest error detection capability at lowest cost

- OMV VEHICLE AND REQUIREMENTS SUMMARY
- ADA AND THE OMV PROJECT
- THE TLD ADA COMPILER
- ADA FEATURES UNDESIRABLE IN OMV REAL-TIME APPLICATIONS
- TASKING INEFFICIENCIES
- FUTURE OF ADA AND THE OMV

Ada and the OMV Project

- OMV PROGRAM DIRECTED TO USE THE Ada PROGRAMMING LANGUAGE IN FLIGHT AND GROUND SOFTWARE
- MIL-STD - 1750A ARCHITECTURE SELECTED FOR THE ON-BOARD FLIGHT PROCESSOR
- TLD SYSTEMS, LTD. Ada COMPILER SELECTED FOR 1750A CODE GENERATION
- UTILIZED VAX 8650, TLD Ada, AND VAX Ada FOR PROTOTYPE DEVELOPMENT

The TLD Ada Compiler

- PERFORMS INTELLIGENT OPTIMIZATION OF SOURCE CODE IN PRODUCING EFFICIENT OBJECT CODE
- THE MOST FREQUENTLY USED LANGUAGE FEATURES AVERAGED 1:5 ADA TO MACHINE CODE EXPANSION RATIO
- EACH LINE OF ADA CODE AVERAGED 7.5 WORDS OF MEMORY AND TOOK 10.5 MICROSECONDS TO EXECUTE
- THE MORE ADVANCED FEATURES OF ADA TENDED TO BE TOO INEFFICIENT FOR USE IN REAL-TIME SYSTEMS

Ada Features Undesirable in OMV Real-Time Applications

SMALL INEFFICIENCIES :

- VARIANT RECORDS
- IF STATEMENTS WITH COMPOUND CONDITIONS
- PRIVATE TYPES AS FORMAL PARAMETERS IN GENERICS
- DATA STRUCTURES THAT USE DYNAMIC MEMORY
- DECLARATION OF ARRAYS WITH INITIAL VALUES

LARGE INEFFICIENCIES :

- TASKING

Tasking Inefficiencies

- LANGUAGE REFERENCE MANUAL (LRM) PROVIDES FOR ONLY FIXED PRIORITY LEVELS FOR TASKS
- ENTRY QUEUES ARE FIRST IN, FIRST OUT (FIFO) ONLY
- NEED THE ABILITY TO SPECIFY A TASK AS NON-PREEMPTIBLE BY OTHER TASKS
- TASKING IMPOSES SERIOUS SIZING AND TIMING IMPACTS
- LARGE OVERHEAD IN TASK ELABORATION, INITIALIZATION, AND ACTIVATION

Future of Ada and the OMV

- THE OMV FLIGHT SOFTWARE WILL USE TRADITIONAL EXECUTIVE ARCHITECTURE EXCLUDING TASKING
- THE FLIGHT SOFTWARE WILL AVOID THE USE OF DYNAMIC MEMORY AND LIMIT THE USE OF GENERICS
- THE GROUND SOFTWARE WILL USE VAX Ada ON A VAX SYSTEM

AERO-ASSIST FLIGHT EXPERIMENT (AFE)

- CURRENTLY AN IN-HOUSE PROJECT WITH INDUSTRY BRIEFING SCHEDULED FOR DECEMBER 15, 1988

- 2 OBCs : EXPERIMENT AND GN&C WITH A 1553 SHARED BUS

- SRR SCHEDULED FOR FEBRUARY 1989

- DEVELOPING A SOFTWARE SIMULATION LAB BASED ON A VAX 8650 WITH A 68020 CROSS COMPILER

- GN&C FLIGHT SOFTWARE DESIGN IN-HOUSE USING DEC Ada

- COMPILE AND EXECUTE THE PACKAGE SPECIFICATIONS

AERO-ASSIST FLIGHT EXPERIMENT (AFE)

- COMMENTS FROM LEAD SOFTWARE ENGINEER FOR AFE :
  - LIMITED Ada BENCHMARK PROGRAMS
  - LIMITED COMPILER VENDORS
  - SLOW Ada COMPILATION TIMES
  - INEFFICIENCY OF TASKING
  - GROUND SOFTWARE MAY NOT BE IN Ada

SECURE SHUTTLE DATA SYSTEM (SSDS)

- EXISTING VAX FORTRAN PROGRAM (7000 SLOC) RUNNING ON A VAX 11/780

- PROGRAM REDESIGN TO HANDLE 3 OPERATIONAL DOWNLINK STREAMS FOR STS-27 MISSION

- Ada CHOSEN AS THE LANGUAGE FOR THE NEW PROGRAM :

  -- TO BE CONSISTENT WITH DoD

  -- MSFC NEEDED Ada REAL-TIME EXPERIENCE

- NEW Ada PROGRAM RUNNING ON A PERKIN-ELMER 3244 USING CONCURRENT COMPUTER'S Ada COMPILER (15000 SLOC)

SECURE SHUTTLE DATA SYSTEM (SSDS)

- REASONS FOR DELAY IN DELIVERY OF SYSTEM :
  - -- LACK OF REAL-TIME Ada EXPERIENCE
  - -- INSUFFICIENT FAMILIARITY WITH HOST OPERATING SYSTEM AND SERVICES
  - -- BUGS AND PERFORMANCE PROBLEMS WITH THE INITIAL VERSION OF THE COMPILER
  - -- NOT ENOUGH TIME SPENT IN THE DESIGN PHASE
  - -- NOT IN THE "Ada MINDSET"

SECURE SHUTTLE DATA SYSTEM (SSDS)

- PROBLEMS WITH TASKING :

  -- CONCEIVED TASKS IN Ada AS BEING INDIVIDUAL PROCESSES

  -- IN REALITY, TASKING WAS SUPPORTED AS A SINGLE PROCESS WITH ROUND-ROBIN SCHEDULING

  -- THE COMPILER DID NOT SUPPORT THE PRAGMA PRIORITY

  -- DID USE TASKING AND RENDEZVOUS

  -- USED THE DELAY STATEMENT TO IMPLEMENT PRIORITIES

JPL

RWP

REAL-TIME WEATHER PROCESSOR (RWP) PROJECT

ADA EXPERIENCE AT PDR

PATRICIA M. MOLKO
ROBERT E. LOESH

1 DECEMBER 1988

AGENDA

- WHAT IS THE RWP SYSTEM
- REASONS FOR CHOOSING ADA AND ITS IMPACT
- ADA RISKS: ASSESSMENT AND CONTROL
- ADA TRAINING APPROACH
- DEVELOPMENT APPROACH
- LESSONS LEARNED/RECOMMENDATIONS

WHAT IS THE RWP SYSTEM?

- SPONSOR: FEDERAL AVIATION ADMINISTRATION (FAA)
- PROTOTYPE DEVELOPMENT; EVENTUALLY PART OF NATIONAL AIRSPACE SYSTEM UPGRADE
- RWP INITIATED OCTOBER 1987 AS A RESULT OF RESCOPING CENTRAL WEATHER PROCESSOR (CWP) PROJECT:
  - AT TIME OF RESCOPING HAD STARTED DETAILED DESIGN
  - 3 TIMES SIZE OF RWP
  - C LANGUAGE AND TAILORED DOD-STD-2167
- 1 RWP SYSTEM AT 21 OF 23 AREA CONTROL FACILITIES; 7 EXTERNAL INTERFACES

JPL

WHAT IS THE RWP SYSTEM? (CONT'D)

RWP

- RWP WILL RECEIVE WEATHER DATA AND PROVIDE AUTOMATIC DISSEMINATION OF PERTINENT WEATHER INFORMATION TO AIR TRAFFIC CONTROLLERS AND METEOROLOGISTS

- CURRENTLY IN DETAILED DESIGN PHASE; SYSTEM TO BE DELIVERED TO FAA AUGUST 1990

- S/W INTENSIVE; H/W OFF-THE-SHELF

  - 1 COMPUTER S/W CONFIGURATION ITEM

    - DEVELOPED BY JPL: 72,000 SLOC (ADA)
    - COMMERCIAL OFF-THE-SHELF: (COMMUNICATIONS PROTOCOLS) 133,000 SLOC (C)

    TOTAL: 205,000 SLOC

  - ADA, DOD-STD-2167, REVISION A: TAILORED

WHAT IS THE RWP SYSTEM? (CONT'D)

- o DISTRIBUTED H/W ARCHITECTURE
  - o 10 MICRO VAX IIs, 3 MICRO VAX 3600s, 1 MICRO VAX 3200
  - o VAXELN AND VAX/VMS OPERATING SYSTEMS, DECNET, ISO PROTOCOLS
- o DEVELOPMENT ENVIRONMENT
  - o VAX 8600 AND IBM PC/ATs
  - o 2 MICRO VAX WORK STATIONS FOR S/W DEVELOPERS
  - o TARGET SYSTEM (MIRROR IMAGE OF PROTOTYPE)
  - o JPL DEVELOPED TEST DATA GENERATOR SYSTEM TO SIMULATE EXTERNAL INTERFACES
  - o DEC VAX ADA TOOL SET
  - o ADAGEN, ADAMAT, DBASE III, YOURDON TOOLSET

REASONS FOR CHOOSING ADA AND ITS IMPACT

- FAA FAVORED USE OF ADA FOR NEW PROJECTS: MITRE STUDY (4/87) RECOMMENDED ADA FOR FAA'S ADVANCED AUTOMATION SYSTEM
- JPL'S TOP MANAGEMENT INTERESTED IN ADA
- PORTABILITY OF RWP PROTOTYPE TO FIELDABLE SYSTEMS ENHANCED
- PROJECT MANAGEMENT AND STAFF INTERESTED IN USING ADA (SOME MIXED REACTION)
- ADA'S FEATURES PROMOTE SOUND S/W ENGINEERING PRACTICES

JPL

REASONS FOR CHOOSING ADA AND ITS IMPACT (CONT'D)

RWP

- o SCHEDULE IMPACT
  - o 2 MONTHS ADDED TO SCHEDULE COMPLETION DATE
  - o INCREASED PRELIMINARY DESIGN PHASE BY 33%
  - o INCREASED DETAILED DESIGN PHASE BY 10%
- o COST IMPACT
  - o INCREASED COST BY:
    - PLANNED DEVELOPMENT: 8% ($2.5M)
    - WITH ADDED RESERVE: 10% ($ .8M)

ADA RISKS: ASSESSMENT AND CONTROL

- o PERFORMANCE RISKS
  - o SET OF RWP ADA BENCHMARKS RUN ON DEC SHOWED ADEQUATE PERFORMANCE MARGIN
- o SYSTEM SIZE RISKS
  - o RWP IS A MEDIUM-SIZE SYSTEM (72,000 SLOC DEVELOPED) SO COMPILER ADEQUACY NOT A CONCERN
- o PERSONNEL/MANAGEMENT RISKS
  - o 3-MONTH UP FRONT TRAINING PERIOD FOR ALL PERSONNEL PLUS CASE STUDY WORKSHOP FOR S/W DEVELOPMENT STAFF
  - o RWP MANAGEMENT STAFF HAS AN AVERAGE OF 15 YEARS MANAGEMENT EXPERIENCE; ATTENDED ADA TECHNICAL AND MANAGEMENT SEMINARS

JPL

ADA RISKS: ASSESSMENT AND CONTROL (CONT'D)

RWP

- COMPUTING ENVIRONMENT RISKS
  - DEC VAX 8600 DEVELOPMENT ENVIRONMENT, MATURE
  - DEC TARGET ENVIRONMENT, MATURE
  - ALL COMMUNICATIONS BETWEEN PROCESSORS VIA DECNET, MATURE
- SCHEDULE AND COST RISKS
  - USED INDUSTRY AND NASA EXPERIENCE FROM SEVERAL ADA PROJECTS
  - OVERALL APPROACH FOR RISK CONTROL: RAPID PROTOTYPING, INCREMENTAL DEVELOPMENT, FAGAN INSPECTION METHODOLOGY
  - OBTAIN FAA MANAGEMENT FLEXIBILITY AT OUTSET

JPL

ADA TRAINING APPROACH

RWP

- NO SIGNIFICANT ADA EXPERIENCE

- TWO ADA EXPERTS HIRED TO ASSIST IN:
  - TRAINING
  - S/W METHODOLOGIES AND PROCEDURES DEVELOPMENT

- FIRST TRAINING SESSION
  - 25 PARTICIPANTS: S/W DEVELOPERS AND SOFTWARE PRODUCT ASSURANCE STAFF
  - 3 DAYS VAX VMS ORIENTATION
  - 3 WEEKS FORMAL ADA TRAINING
  - 2 DAYS OF OBJECT ORIENTED DESIGN (OOD) FORMAL TRAINING PLUS GSFC BRIEFING ON GENERAL OOD METHODOLOGY (GOOD)

JPL

ADA TRAINING APPROACH

RWP

- 7 WEEKS OF ADA TRAINING CASE STUDY WORK (RWP APPLICATIONS)

o SECOND TRAINING SESSION

- 22 PARTICIPANTS INCLUDING FAA STAFF; SYSTEM ENGINEERING, TEST & OPERATIONS, PROJECT MANAGEMENT AND OTHER SUPPORT STAFF
- 2-1/2 DAY VAX VMS ORIENTATION
- 2 WEEKS FORMAL ADA TRAINING
- 2 DAYS OBJECT ORIENTED DESIGN FORMAL TRAINING (NOT FAA; 1/2 OF JPL)
- NOT PARTICIPATING IN ADA TRAINING CASE STUDY ACTIVITY

JPL

DEVELOPMENT APPROACH

RWP

- ONE CSCI, ONE HWCI
- INCREMENTAL DEVELOPMENT
  - ONE PDR
  - 3 CDRS FOR EACH OF 3 SYSTEM BUILDS, THE FIRST CONSISTING OF THE SYSTEM BACKBONE, THEN ADDING APPLICATIONS
- REWROTE CWP SYSTEM AND SOFTWARE REQUIREMENTS FOR RWP; USED YOURDON AND WARD/MELLOR METHODOLOGY
- USED GSFC "GOOD" METHODOLOGY WITH SOME TAILORING FOR PRELIMINARY DESIGN
- ADA USED AS A PDL
  - 5% OF CODE AT PDR

JPL

DEVELOPMENT APPROACH

RWP

- o 15% OF CODE AT CDR
- o USED FAGAN INSPECTION METHODOLOGY, LEAD BY S/W PRODUCT ASSURANCE STAFF
- o TAILORING OF SOFTWARE DESIGN DOCUMENTS (SDD) FOR USE WITH ADA

JPL

LESSONS LEARNED/RECOMMENDATIONS

RWP

- DOCUMENT RISKS UP RMP; MAKE SURE SPONSOR UNDERSTANDS THEM
- BEST TO HAVE TRAINING AND METHODOLOGY IN PLACE PRIOR TO STARTING
- RWP TRAINING APPROACH WORKED WELL
  - VERIFIED METHODOLOGY DURING TRAINING
  - PROVIDED PROJECT-SPECIFIC CASE STUDIES
  - PROVIDED ADA EXPERTS TO ASSIST STAFF
- CONTROL RISKS BY SETTING SMALL INTERMEDIATE MILESTONES AND CLOSELY MONITOR PROGRESS
- MAY NEED TO INCREASE PRELIMINARY DESIGN PHASE BY 40-45%; DETAILED DESIGN PHASE BY 25%

JPL

LESSONS LEARNED/RECOMMENDATIONS (CONT'D)

RWP

- OBJECT-ORIENTED DESIGN REQUIRES SIGNIFICANT TRAINING; MORE CASE STUDY WORK RECOMMENDED
- DESIGN PHASE METHODOLOGY DID NOT SUPPORT "INSPECTIONS-AS-YOU-GO" METHODOLOGY VERY WELL
- <u>CAVEAT EMPTOR</u> REGARDING BUYING COMMERCIAL OFF-THE-SHELF SOFTWARE

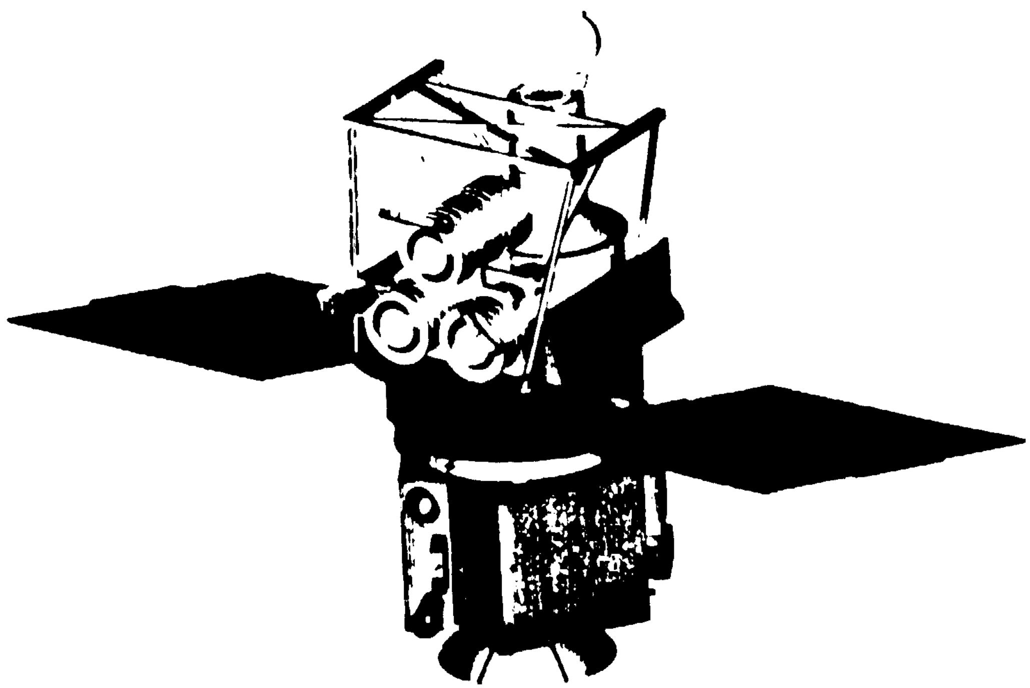
Session 2: APPLICATIONS

1. Barbara Scott, NASA/GSFC
2. Kathy Schubert, NASA/LeRC
3. Brandon Rigney and Cora Carmody, PRC
4. David Badal, Lockheed

NASA

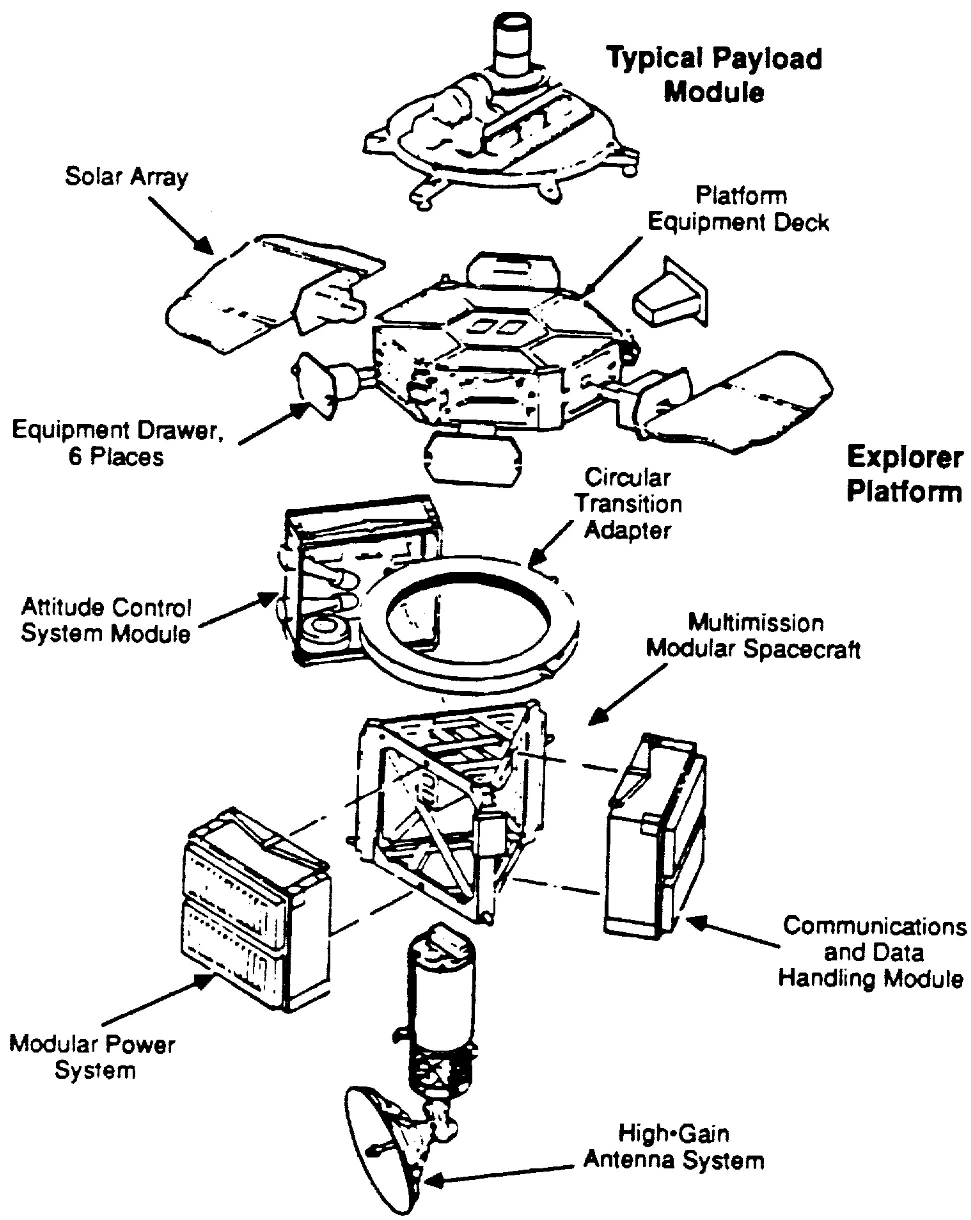
Explorer PLatform ADA Flight Software

GSFC



Barbara Scott
Satellite Servicing Project
Goddard Space Flight Center

Explorer Platform Configuration

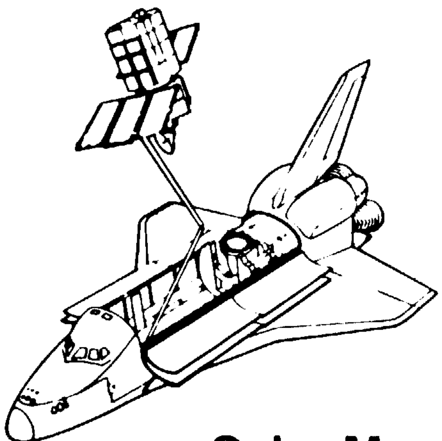


Explorer PLatform Spacecraft

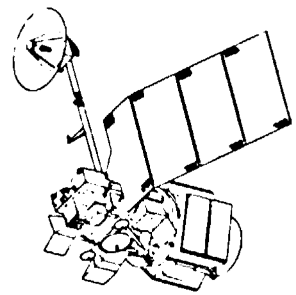
NASA

MMS Spacecraft Software

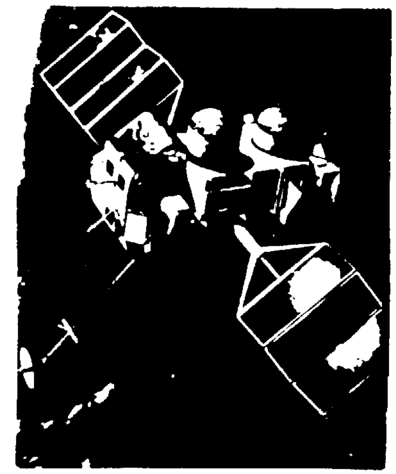
GSFC



Solar Max

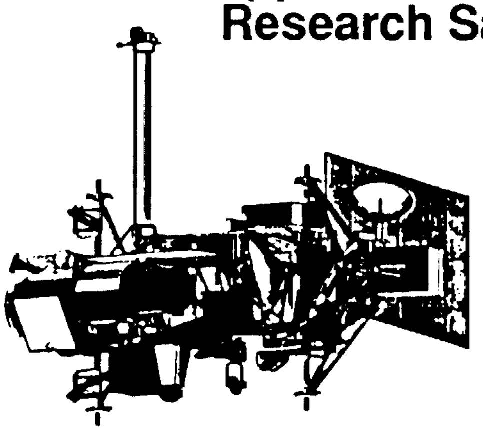


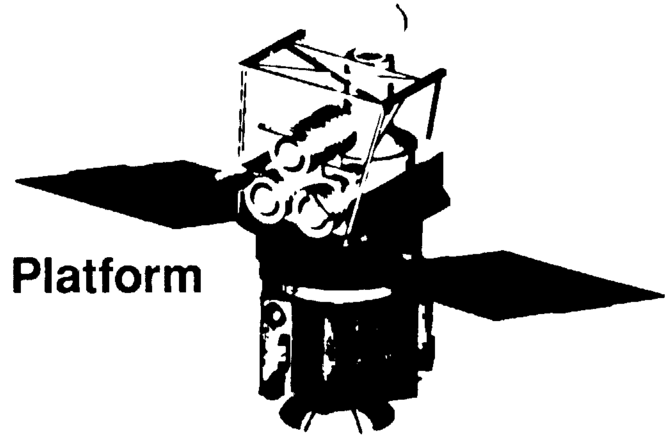
Landsat



Gamma Ray
Observatory

Upper Atmosphere
Research Satellite





Explorer Platform

NASA

RISKS AND RESULTING GROUND RULES

GSFC

RISKS

- NEW HIGH LEVEL LANGUAGE
- NEW CROSS - COMPILER
- NEW HOST COMPUTER

REACTIONS

- NSSC - I - MASTER, CO - PROCESSOR - SLAVE
- CO - PROCESSOR CANNOT INDEPENDENTLY COMMAND THE SPACECRAFT
- CO - PROCESSOR HAS NO MISSION CRITICAL SOFTWARE

NASA

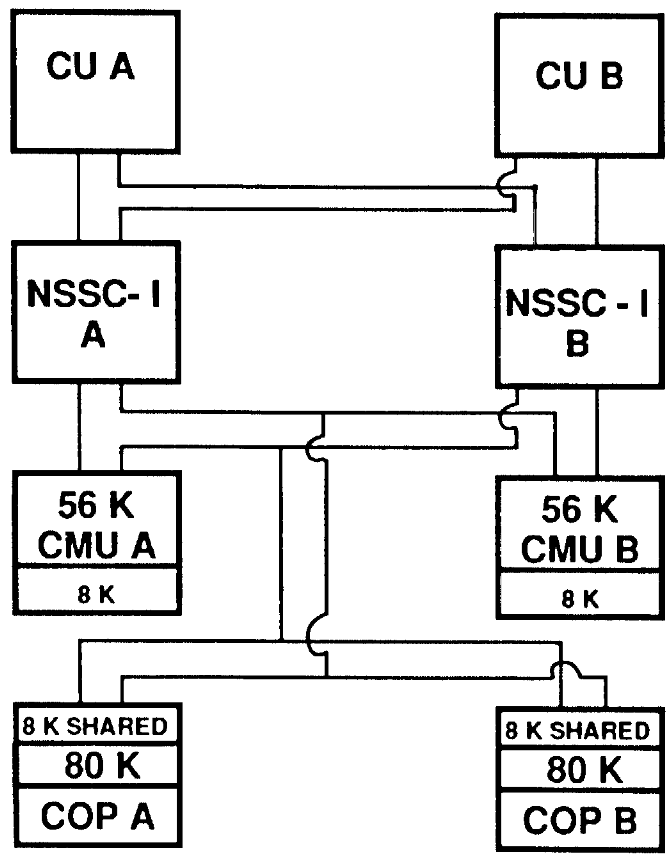
BENEFITS

GSFC

- IMPROVED TECHNOLOGY FOR MULTI - MISSION SPACECRAFT
- LESSONS - LEARNED FOR FUTURE NASA MISSIONS, e. g. OMV, SPACE STATION
- LOWER DEVELOPMENT AND MAINTENANCE COSTS POSSIBLE

EXPLORER PLATFORM ON - BOARD COMPUTER SYSTEM

ARCHITECTURE



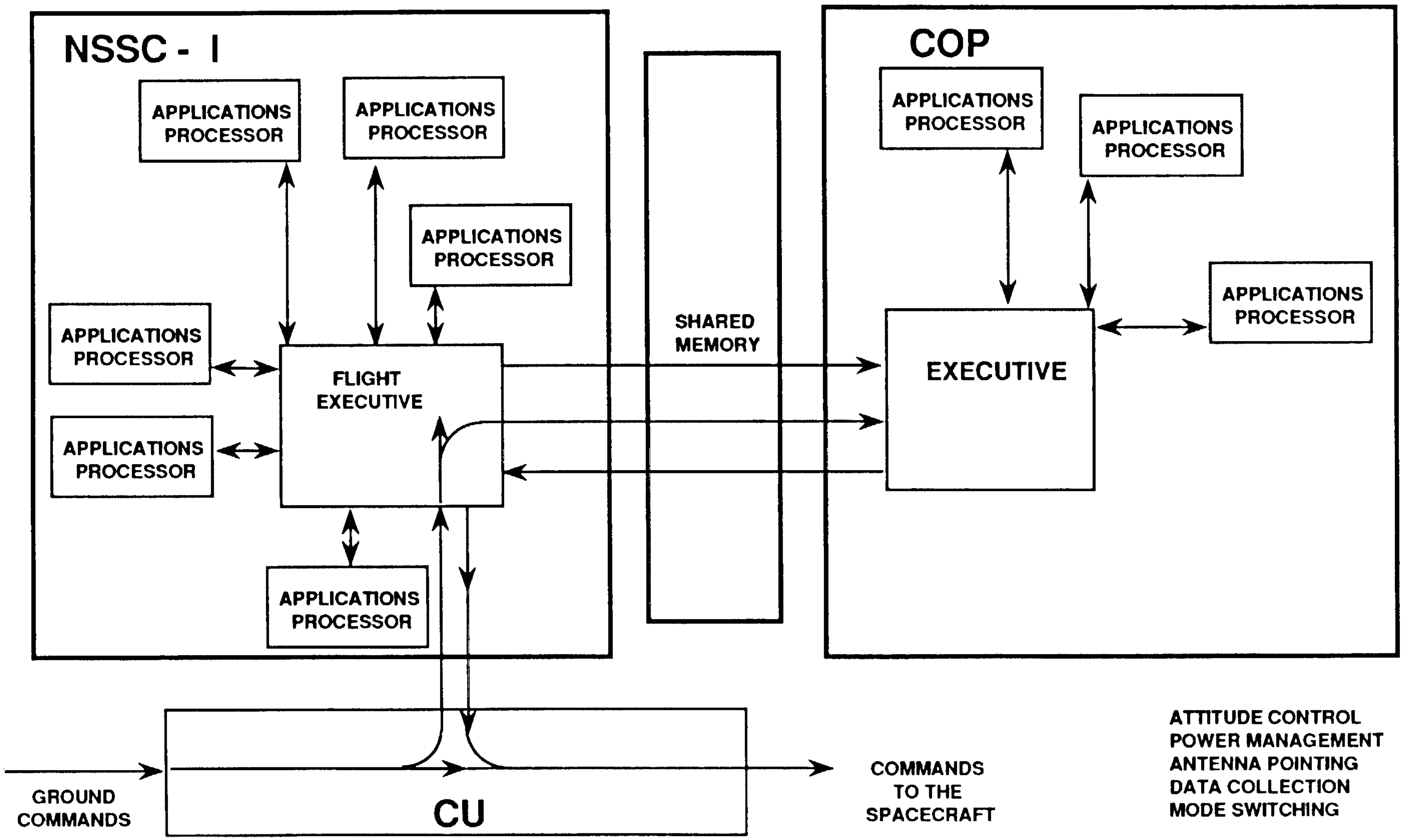
CU - CENTRAL UNIT

NSSC - I - NASA STANDARD SPACECRAFT COMPUTER (PROCESSOR ONLY)

CMU - CORE MEMORY UNIT (NSSC - I MEMORY)

COP - 1750 A CO - PROCESSOR AND MEMORY

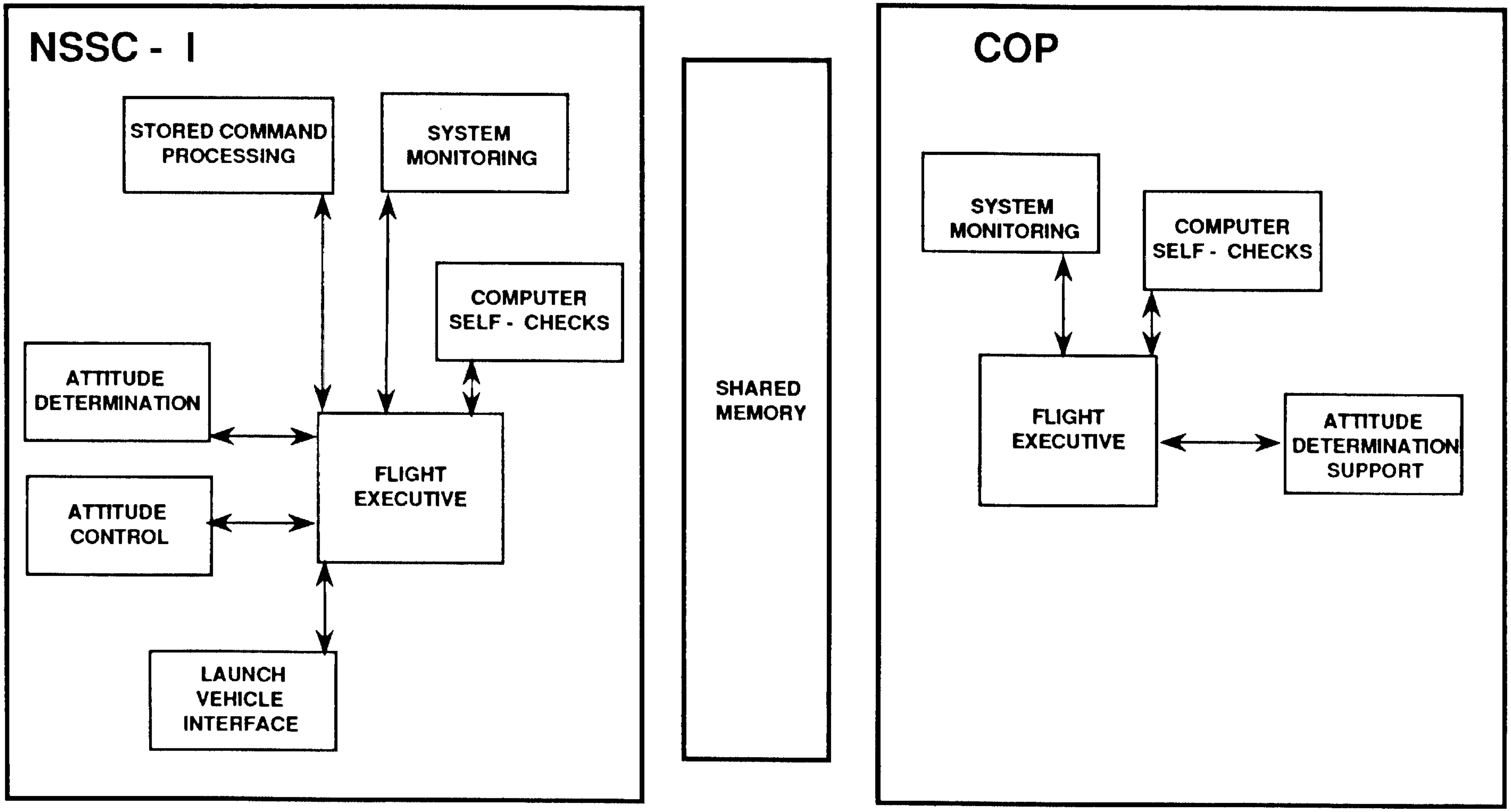
EXPLORER PLATFORM ON - BOARD COMPUTER SYSTEM COMMAND FLOW



NASA

ON - BOARD COMPUTER SOFTWARE FUNCTIONALITY

GSFC



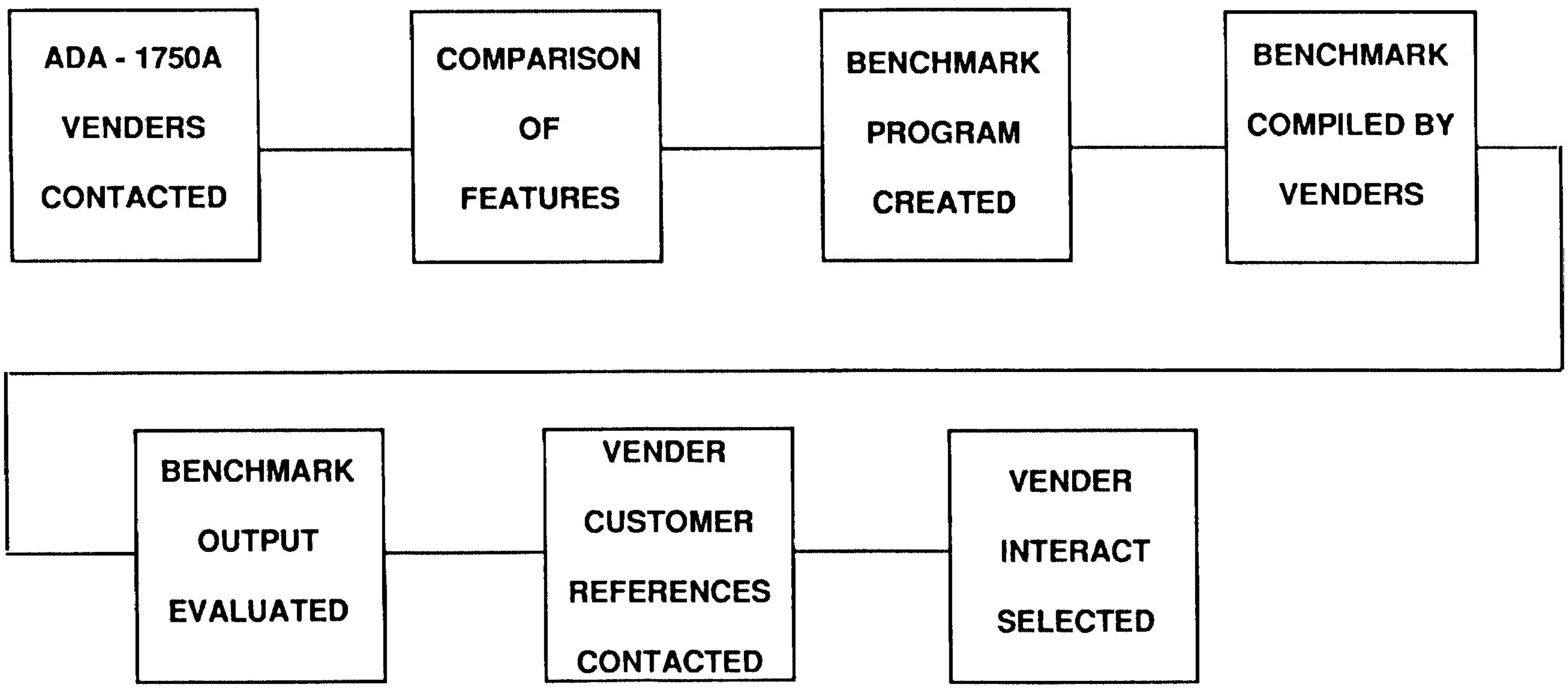
NASA

ADA ISSUES AND CONCERNS:

GSFC

ISSUE	CONCERN
1. COMPILER SELECTION	VALIDATED? MATURE?
2. COMPILER EFFICIENCY	TIMING, SIZE OF TARGET CODE
3. ADA KERNAL - RUN TIME SYSTEM	SIZE? TIMING? MODIFIABLE?
4. MEMORY MANAGEMENT SUPPORT	PAGING SCHEME
5. TOOL SET SUPPORT	FLEXIBLE LINKER? HP - 64000 MDS UTILITY ?
6. IMPLEMENTATION DEPENDENT FEATURES	IN - LINE ASSEMBLY CODE?
	PRAGMA PRIORITY?
7. LICENSING AGREEMENT	CHANGING "OWNERSHIP"

NASA ISSUE I COMPILER SELECTION



ISSUE 2 COMPILER EFFICIENCY

SOFTWARE MODULE	1750 MEMORY WORDS — RUN TIME CHECKS ON	1750 MEMORY WORDS — RUN TIME CHECKS OFF	
Ada Run Time Executive (rel. 3.1)	6385 (3810,239,2336)		
COP Flight Executive	17585 (12303,399,4883)		
Instruction Test	1011 (816,6,189)		
Unused Memory Test	859 (849,10,0)		
Exclusive-OR Test	615 (601,14,0)		
Memory Monitor	173 (167,6,0)		
COP OK	141 (131,10,0)		
SUBTOTAL	26769	22580	
APPLICATIONS PROCESSORS			
Update Filter	17946 (16371,144,1541)	6180 (4695,54,1431)	
Math Library	3765 (3455,229,81)	3018 (2780,157,81)	
Statistics Monitor	8489 (6082,202,2205)	6969 (4562,202,2205)	
TOTAL	56969	38747	32% SAVINGS
DATA AREAS			
System Heap/ Stack	12288 (0,0,12288)	12288 (0,0,12288)	
Star Catalog	2100 (0,0,2100)	2100 (0,0,2100)	

UPDATE FILTER CONTROL PROCESSOR

SOFTWARE MODULE	ADA LOC	1750A MEMORY WORDS		RATIO WORDS PER LOC	
		RUN TIME CHECKS		RUN TIME CHECKS	
		ON	OFF	ON	OFF
UF_DATA_DEF (spec)	52	1437 (32, 2, 1403)	1434 (29, 2, 1403)	27.6:1	27.6:1
UF_DATA_TYPES (spec)	16	30 (28, 2, 0)	27 (25, 2, 0)	1.9:1	1.7:1
UF_ONE (spec)	6	36 (32, 2, 2)	33 (29, 2, 2)	6.0:1	5.5:1
UF_ONE (body)	27	460 (454, 6, 0)	178 (176, 2, 0)	17 :1	6.6:1
UF_TWO (spec)	25	57 (46, 2, 9)	54 (43, 2, 9)	2.3:1	2.2:1
UF_TWO (body)	223	4877 (4827, 50, 0)	2050 (2036, 14, 0)	22 :1	9.2:1
UF_THREE (spec)	12	48 (40, 2, 6)	45 (37, 2, 6)	4.0:1	3.7:1
UF_THREE (body)	109	4519 (4491, 28, 0)	879 (865, 14, 0)	41 :1	8.1:1
UF_FOUR (spec)	9	42 (36, 2, 4)	39 (33, 2, 4)	4.7:1	4.3:1
UF_FOUR (body)	96	4958 (4930, 28, 0)	910 (904, 6, 0)	52 :1	9.5:1
UF_TASK (body)	18	78 (78, 0, 0)	51 (51, 0, 0)	4.3:1	2.8:1
FLOAT_UTILITY (spec)	19	51 (42, 2, 7)	48 (39, 2, 7)	2.7:1	2.5:1
FLOAT_UTILITY (body)	41	1353 (1335, 18, 0)	432 (428, 4, 0)	33 :1	10.5:1
TOTAL	653	17946 (16371, 144, 1431)	6180 (4695, 54, 1431)	27.5:1	9.5:1

KEY: (code, literals, data)

COMPILER COMPARISON

InterACT VS TARTAN

		SMP_DATA	SMP	SMP_DATA_SPEC	SMP_TDATA_SPEC	
InterACT - ALL RUN TIME CHECKS ON	LOC:	99	647	247	73	
	CODE:	1207	3399	28	1094	
	DATA:	74	42	1490	791	
	TOTAL:	**1281**	**3441**	**1518**	**1885**	=8125
TARTAN - ALL RUN TIME CHECKS ON	LOC:	100	627	246	74	
	CODE:	1076	2319	155	969	
	DATA:	2331	26	1648	740	
	TOTAL:	**3407**	**2345**	**1803**	**1709**	=9264
TARTAN - /NO_ENUMERATION	CODE:	1056	2319	135	965	
	DATA:	2185	26	1502	724	
	TOTAL:	**3241**	**2345**	**1637**	**1689**	
TARTAN - /NO_ENUMERATION /NO_CONSTRAINT	CODE:	1056	1134	135	802	
	DATA:	2185	26	1502	724	
	TOTAL:	**3241**	**1160**	**1637**	**1526**	
TARTAN - /NO_ENUMERATION /NO_CONSTRAINT /NO_STACK_CHECK	CODE:	1044	1109	126	731	
	DATA:	2185	26	1502	724	
	TOTAL:	**3229**	**1135**	**1628**	**1455**	

HEAP / STACK

HEAP - A COLLECTION OF TASK STACKS PLUS A GENERAL USER AREA

- INCLUDE A KERNAL MAIN TASK STACK - SIZE 2K
- ALL OTHER STACKS MUST BE THE SAME SIZE

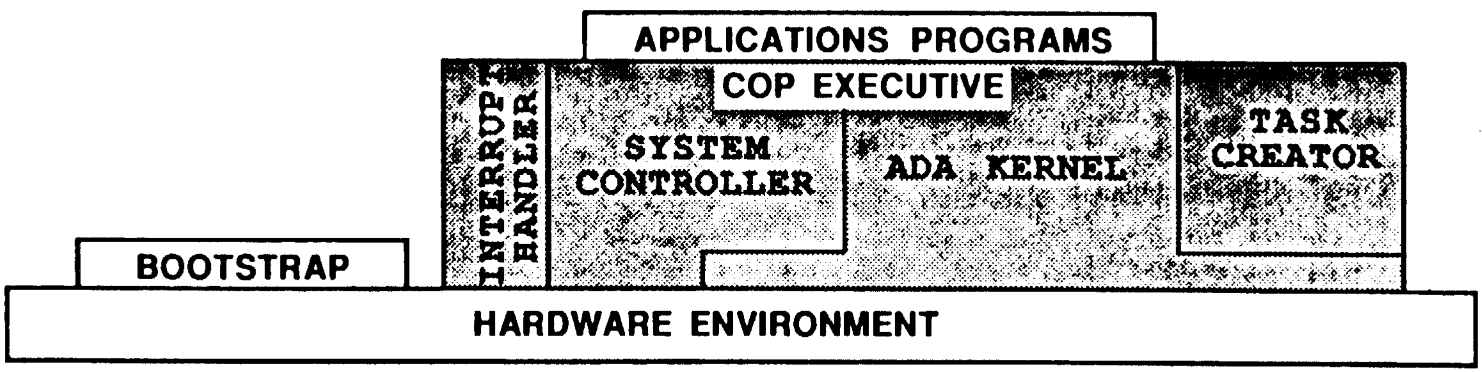
  I.E. THE SIZE OF THE LARGEST STACK NEEDED

WHAT SIZE SHOULD OUR TASK STACKS BE?

- CREATED A LOAD MODULE WITH 5 REAL AND 26 "DUMMY" TASKS
- ORIGINAL STACK SIZE OF 512 WORDS FOR EACH TASK
  - COMPILED AND EXECUTED SUCCESSFULLY
- TRIED STACK SIZE OF 100 WORDS FOR EACH TASK
  - COMPILED BUT DID NOT EXECUTE
- TRIED STACK SIZE OF 256 WORDS FOR EACH TASK
  - COMPILED BUT DID NOT EXECUTE
- RETURNED TO STACK SIZE OF 512 WORDS
- WILL DETERMINE SMALLEST STACK SIZE AFTER LARGEST TASK

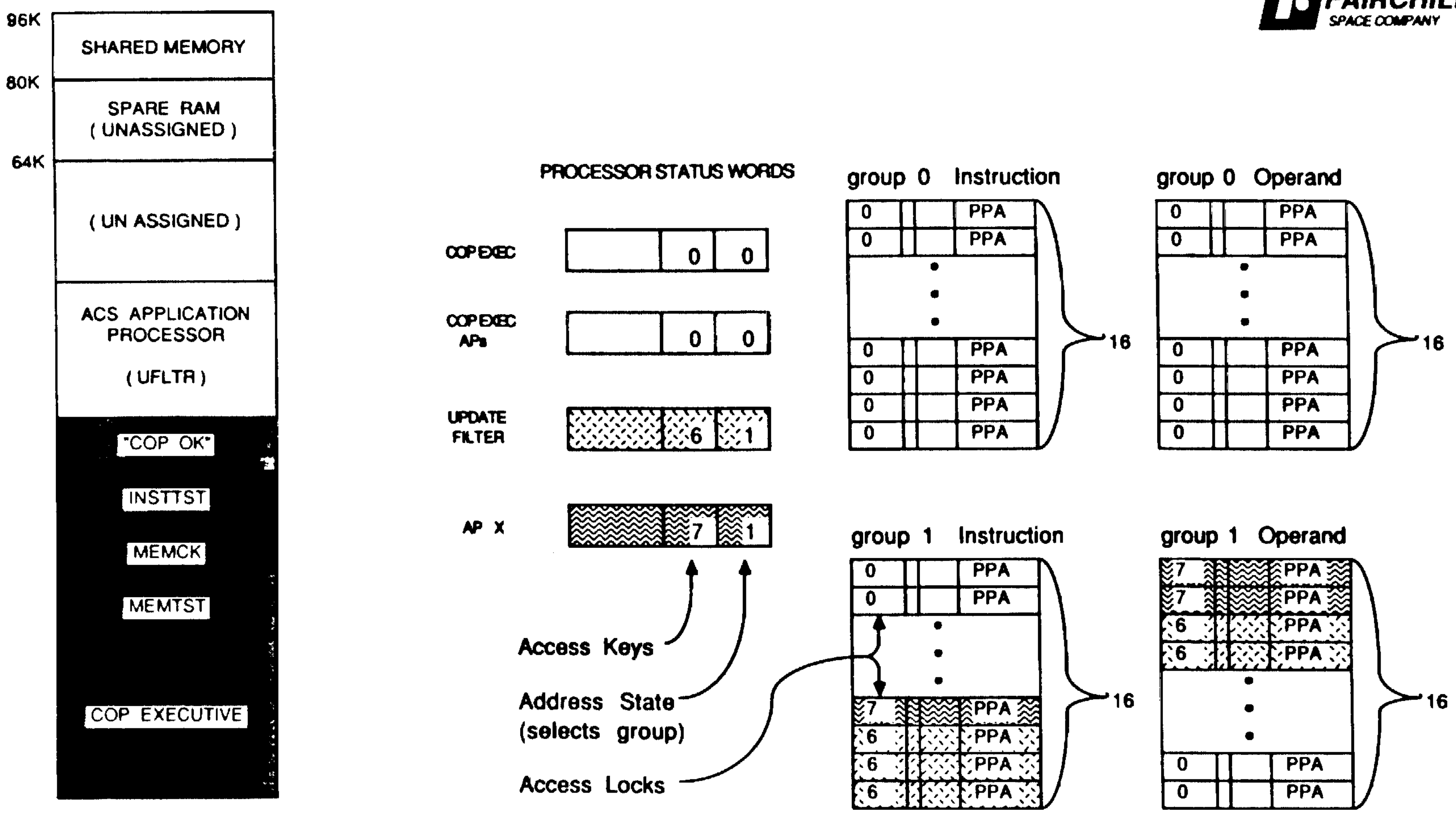
  (UPDATE FILTER) IS INCLUDED IN LOAD MODULE

ISSUE 3 ADA RUN TIME KERNEL



ORIGINAL PAGE IS
OF POOR QUALITY

MEMORY MANAGEMENT SUPPORT



NASA

ISSUE 5 TOOL SET

GSFC

- Ada to 1750A Cross Compiler
- 1750A Assembler
- Compiled Code Library Manager
- Linker
- HP - 64000 Microprocessor Development System (MDS)
  Conversion Utility

Implementation Dependent Features

WE NEED:

Long Float
Preemptive Scheduling
Machine Code Insertion
Interrupt Entries
Data Representation

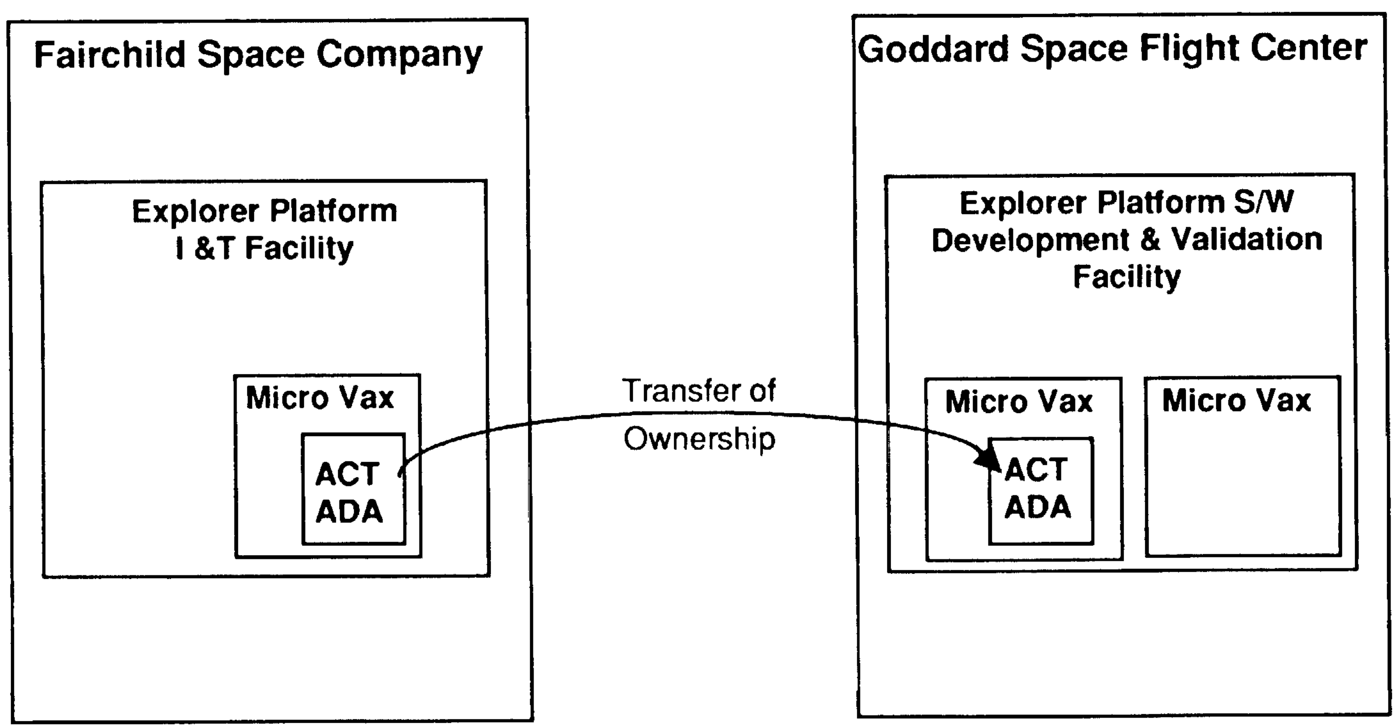
- Bit Level
- Data Address Clauses
- Task Length Clauses
- Enumeration

Predefined Pragmas

- Interface
- List
- Page
- Priority
- Supress

NASA

ISSUE 7 Licensing Agreement



CODE COMPARISON

FORTRAN PDL LOC - 54

ADA PDL LOC - 23

SPEC - 9

BODY - 14

NSSC - I MEMORY WORDS - 368 PLUS 260 IN SUBROUTINES

(CODE ONLY)

1750A MEMORY WORDS - 681

(CODE ONLY)

FORTRAN PDL EXAMPLE

```
SUBROUTINE STATE TRANSITION

X1(1) = FLOAT (WXC,1,-5)     ** CONVERT INPUTS TO FLOATING POINT
X1(2) = FLOAT (WYC,1,-5)
  :              :
  :              :
X1(9) = FLOAT (VRZ,1,-65)

TPS = TP                     ** SAVE THE CURRENT PROPAGATION INTERVAL

X2(1) = HALF (TP*TP)         ** COMPUTE INTERMEDIATE VARIABLES
X2(2) = (X2(1)*TP)/3.0       ** NEEDED FOR THE MATRICES
X2(3) = X1(1) * X1(1)        ** COMPUTATIONS
X2(4) = X1(2) * X1(2)
X2(5) = X1(3) * X1(3)
X2(6) = X1(1) * TP
X2(7) = X1(2) * TP
X2(8) = X1(3) * TP
X2(9) = X2(3) + X2(4) + X2(5)
X3(1) = X1(1) * X1(2)
X3(2) = X1(1) * X1(3)
  :       :       :
X3(9) = X1(2) * X3(7)
X4(1) = X1(3) * X3(7)
X4(2) = X1(1) * X2(1)
  :       :       :
X4(7) = X3(3) * X2(2)
                             ** COMPUTE THE ELEMENTS OF THE STATE
                             ** TRANSITION MATRIX
FM11(1) = 1.0 - (X2(4) + X2(5)) * X2(1)
FM11(4) = X2(8) + X3(4) - X4(1)
FM11(7) = -X2(7) + X3(5) + X3(9)
FM11(2) = -X2(8) + X3(4) + X4(1)
FM11(5) = 1.0 - (X2(3) + X2(5)) * X2(1)
FM11(8) = X2(6) + X3(6) - X3(8)
FM11(3) = X2(7) + X3(5) - X3(9)
FM11(6) = -X2(6) + X3(6) + X3(8)
FM11(9) = 1.0 - (X2(3) + X2(4)) * X2(1)
FM12(1) = -TP + (X2(4) + X2(5)) * X2(2)
FM12(4) = -X4(4) - X4(5)
FM12(7) = X4(3) - X4(6)
FM12(2) = X4(4) - X4(5)
FM12(5) = -TP + (X2(3) + X2(5)) * X2(2)
FM12(8) = -X4(2) - X4(7)
FM12(3) = -X4(3) - X4(6)
FM12(6) = X4(2) - X4(7)
FM12(9) = -TP + (X2(3) + X2(4)) * X2(2)

END
```

ADA PDL EXAMPLE

```
package UF_PROC is

  type MATRIX is array (INTEGER range <>, INTEGER range <>) of FLOAT;
   subtype MATRIX3X3 is MATRIX(1..3,1..3);
  X1 : MATRIX3X3;
  X2 : MATRIX3X3;
  X3 : MATRIX3X3;
  X4 : MATRIX3X3;
  StateTra_Blk11 : MATRIX3X3;
  StateTra_Blk12 : MATRIX3X3;

end UF_PROC;
```

```
package body UF_PROC is

  procedure STATE_TRANSITION_MATRIX is

      begin
        -- compute the elements of the state transition matrix
        -- NOTE :       X3(1,1) = propagation time interval
        --              X3(2,3) = SIN(W*Dt)
        --              X3(3,3) = 1 - COS(W*Dt)
        --              X3(2,1) = (1 - COS(W*Dt))/W
        --              X3(3,1) = Dt - (SIN(W*Dt)/W)

      for i in 1..3
      loop
        for j in 1..3
        loop
          StateTra_Blk11(i,j) := +X3(2,3)*X1(i,j)+X3(3,3)*X2(i,j);
          StateTra_Blk12(i,j) := -X4(2,1)*X1(i,j)-X4(3,1)*X2(i,j);
        End loop;
      End loop;

      StateTra_Blk11(1,1) := 1.0 + StateTra_Blk11(1,1);
      StateTra_Blk11(2,2) := 1.0 + StateTra_Blk11(2,2);
      StateTra_Blk11(3,3) := 1.0 + StateTra_Blk11(3,3);

      StateTra_Blk12(1,1) := -X3(1,1) + StateTra_Blk12(1,1);
      StateTra_Blk12(2,2) := -X3(1,1) + StateTra_Blk12(2,2);
      StateTra_Blk12(3,3) := -X3(1,1) + StateTra_Blk12(3,3);

  end STATE_TRANSITION_MATRIX;

end UF_PROC;
```

NSSC-I ASSEMBLY LANGUAGE EXAMPLE

```
*
*                           FM11(1) = 1.0 - (X2(4) + X2(5)) * X2(1)
*
3026                            FLD       X2+9       * FLT. PT. LOAD
3026  060220                    BRM       @FLD       * SUBROUTINE CALL
3027  002367                    DATA      X2+9-@FLD
3030                            FADD      X2+12      * FLT. PT. ADD
      000005                    USE       PROG
3030  060244                    BRM       @FADD      * SUBROUTINE CALL
3031  002346                    DATA      X2+12-@FADD
3032                            FMPY      X2         * FLT. PT. MULTIPLY
      000005                    USE       PROG
3032  060253                    BRM       @FMPY      * SUBROUTINE CALL
3033  002323                    DATA      X2-@FMPY
*
*                           X4(8) = X2(1) * (X2(4) + X2(5))
3034                            FST       X4+21      * INTERMEDIATE
3034  060222                    BRM       @FST       * VALUE
3035  002467                    DATA      X4+21-@FST
3036                            FLD       FONE       * 1.0
3036  060220                    BRM       @FLD       * SUBROUTINE CALL
3037  002502                    DATA      FONE-@FLD
3040                            FSUB      X4+21      * FLT. PT. SUBTRACT
      000005                    USE       PROG
3040  060251                    BRM       @FSUB      * SUBROUTINE CALL
3041  002440                    DATA      X4+21-@FSUB
3042                            FST       FM11       * FINAL RESULT
3042  060222                    BRM       @FST       * SUBROUTINE CALL
3043  002145                    DATA      FM11-@FST


SUBROUTINES CALLED:
  FLD, FADD, FMPY, FST, FSUB
```

THE FIRST COLUMN IS THE NSSC-I MEMORY LOCATION IN OCTAL.

THE SECOND COLUMN IS THE 18-BIT CONTENTS OF THE MEMORY LOCATION.

THE THIRD COLUMN IS THE INSTRUCTION MNEMONIC.

THE FOURTH COLUMN IS THE OPERANDS FOR THE INSTRUCTION.

THE FIFTH COLUMN IS A COMMENT FIELD.

ADA ASSEMBLY LANGUAGE EXAMPLE

```
                ; Source Line 98
0188  8220                     LD4       #1,R2
0189  F420                     CMPRNG    R2,$C$04098$00000
018A  0000
018B  7503                     BZ        %*+3
018C  7EF0                     CALL      @-CP,rts.raise.constraint.error
018D  0000
018E  B220                     SUB4      #1,R2
018F  C020                     MULS      $C$04098$00000+2,R2
0190  0002
0191  50F0                     SETI      #15,rts.unsigned.arith.flag
0192  0000
0193  4A21                     ADD       #$P$04098$00000+72,R2
0194  0048
0195  53F0                     CLRI      #15,rts.unsigned.arith.flag
0196  0000
0197  8132                     MOV       R2,R3
0198  8643                     LDL       0(R3),R4
0199  0000
019A  8620                     LDL       $C$04099$00000,R2
019B  0000
019C  A924                     ADDF      R4,R2
019D  8240                     LD4       #1,R4
019E  F440                     CMPRNG    R4,$C$04098$00000
019F  0000
01A0  7503                     BZ        %*+3
01A1  7EF0                     CALL      @-CP,rts.raise.constraint.error
01A2  0000
01A3  B240                     SUB4      #1,R4
01A4  C040                     MULS      $C$04098$00000+2,R4
01A5  0002
01A6  50F0                     SETI      #15,rts.unsigned.arith.flag
01A7  0000
01A8  4A41                     ADD       #$P$04098$00000+72,R4
01A9  0048
01AA  53F0                     CLRI      #15,rts.unsigned.arith.flag
01AB  0000
01AC  8154                     MOV       R4,R5
01AD  9625                     STL       R2,0(R5)
01AE  0000
```

THE FIRST COLUMN IS THE 1750A CO-PROCESSOR MEMORY LOCATION IN HEX.

THE SECOND COLUMN IS THE 16-BIT CONTENTS OF THE MEMORY LOCATION.

THE THIRD COLUMN IS THE INSTRUCTION MNEMONIC.

THE FOURTH COLUMN IS THE OPERANDS FOR THE INSTRUCTION.

The Evolution of Ada Software to support the Space Station Power Management and Distribution System

Kathy Schubert
NASA Ada Symposium
December 1, 1988

Author Biography

Kathy Schubert is a member of the Space Station Electrical Systems Division at NASA Lewis Research Center, in Cleveland, Ohio. She is currently the Work Package 04, Flight Software Manager, for the Phase C/D Space Station Electrical Power System software. Kathy received a BSEE degree from Ohio Northern University and is currently working on her MSEE at Cleveland State University.

National Aeronautics and Space Administration

Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

I. Introduction

II. Ada Software Development

A. Power Management and Distribution (PMAD) Photovoltaic (PV) Testbed

B. PMAD System Testbed

C. PMAD Integrated Testbed

III. Space Station Electrical Power System

IV. Summary

Introduction

Space Station has chosen Ada as the language of choice for all new Space Station operational software. The embedded applications inherent in the onboard computer architecture made Ada a logical choice, although the lack of Ada experience was a major concern. So, in support of the Electrical Power System (EPS), research and development activities, the Ada Control Program for the Phase I PMAD PV Testbed was initiated. Since that time, the Ada software has evolved from a relatively simple Ada application to a more complex embedded Ada project. The purpose of this presentation is to show the progression of the Ada software applications, the lessons learned, and the problems encountered in applying Ada to a real-time, embedded, power management and distribution (PMAD) system.

National Aeronautics and Space Administration

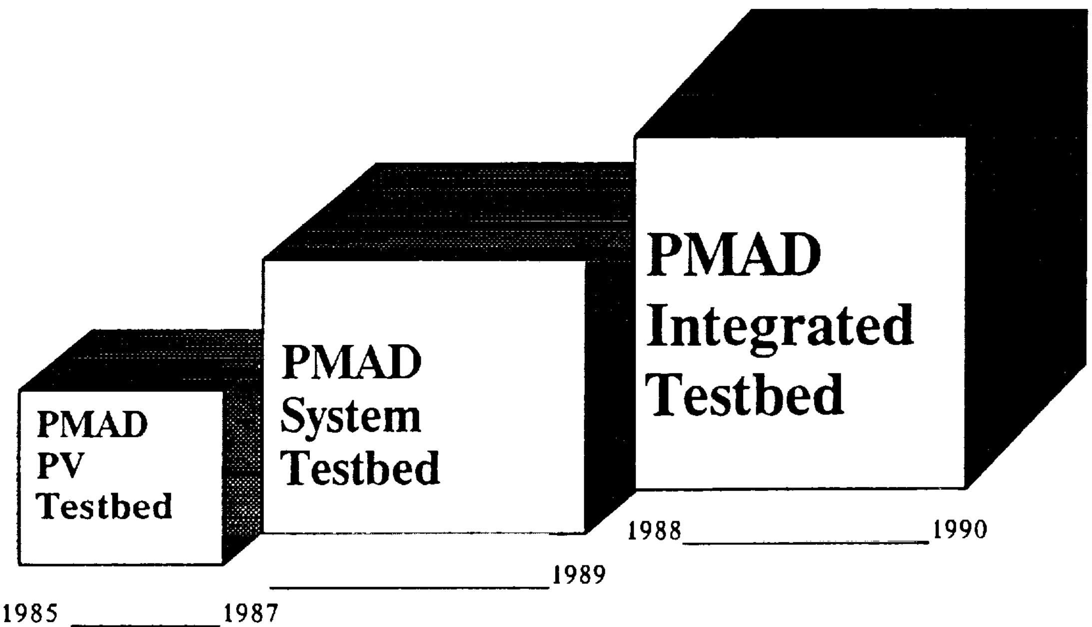
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

Evolution of Ada Software Experience



Ada software experience began with the development of an Ada control program for the Phase I, PMAD PV testbed. The testbed hardware was modeled by Ada simulation software and consisted of a solar array field, a battery bank, a battery charge converter, two load banks, a DC distribution bus, and remote power controllers. This project served as a learning and evaluation phase of Ada for embedded applications. It should be noted that each testbed consists of different system configurations and that each of these represents independent software development efforts. The PMAD System Testbed and the PMAD Integrated Testbed are currently under development and will be discussed briefly. The PMAD PV Testbed software is complete and will serve as the focal point of discussion.

Phase I PMAD PV Testbed Software

- INTEL 8086 based microprocessor environment
- Originally written in FORTRAN
- Utilizes the PAMELA design methodology

The Phase I PMAD PV Testbed hardware consists of a solar array field for power generation, a battery bank for power storage, a DC distribution bus, remote power controllers (RPCs), and a DC to DC charge converter. Simulation software, which characterizes each hardware component, provided the operating environment for the Ada control software. The software runs on the VAX 11/785 under the DEC Ada compiler for initial debugging and is then crossed compiled with the Softech Ada-86 compiler to the iSCB 8086 microprocessor hardware.

The same control and simulation software had previously been written in FORTRAN when this project began. This provided interesting comparisons but resulted in very little documentation and the Ada project started out as a re-coding effort rather than a software development effort. After 10 months into the project, the Ada development team decided to retrofit parts of the software development lifecycle to the project. The testbed hardware requirements were established and the PAMELA design methodology was followed.

National Aeronautics and Space Administration

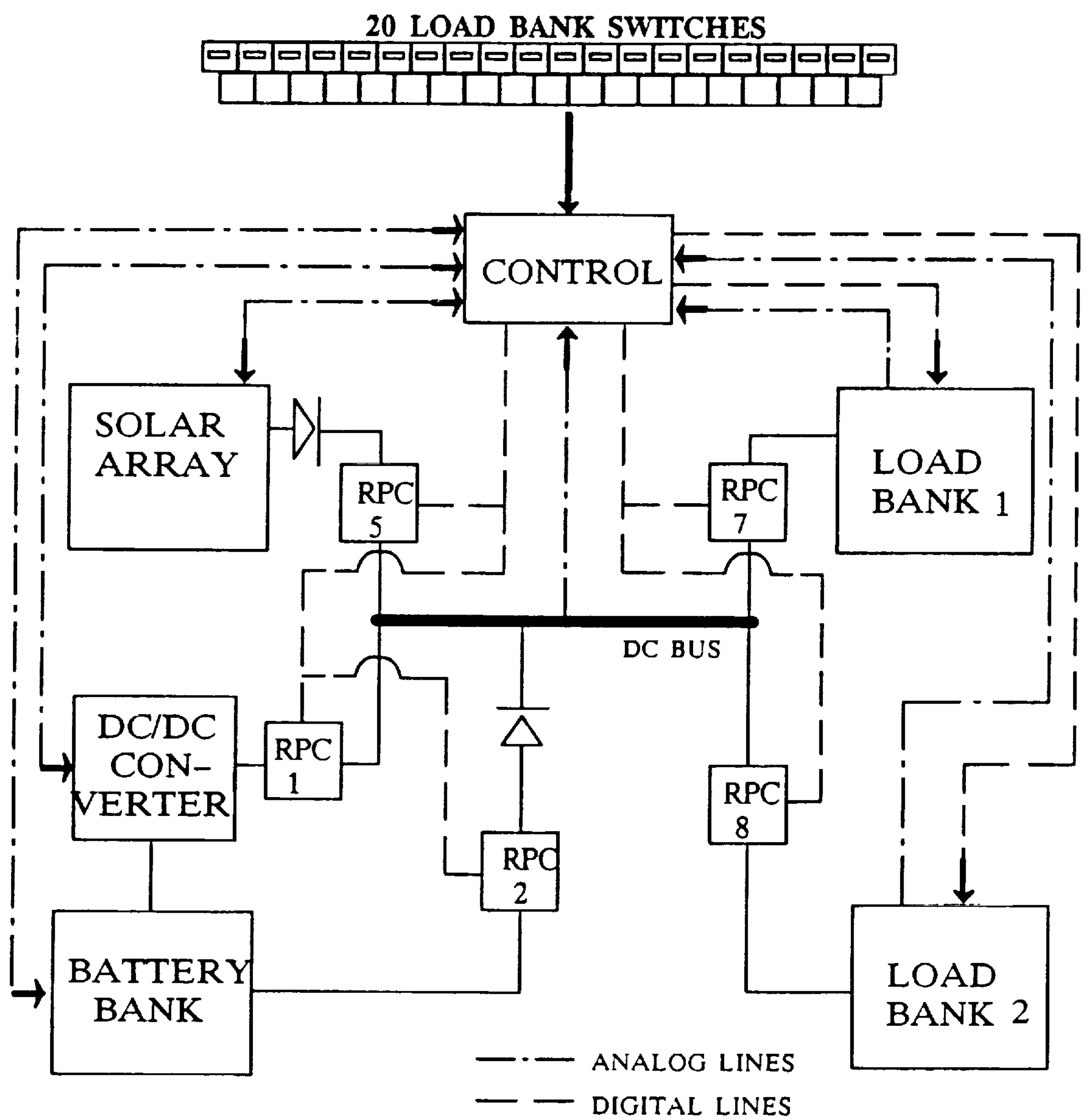
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

PMAD PV Testbed Configuration



Control Software Design

- PAMELA 1 was ideal for designing with Ada tasks but not as well suited to sequential programming
- PAMELA 2 has since been introduced which solves this
- Easy to follow, step by step, **REPEATABLE**, methodology
- Design diagrams are done with a drawing tool

The design phase of the Ada controls program utilizes the PAMELA design methodology. PAMELA is an acronym for Process Abstraction Method for Embedded Large Applications, developed by George Cherry. The Ada control program design consists of a series of graphs which build the program both graphically and textually. The External Object Graph and a simplified Master Subprogram Graph are included here as a top level description of the Controls software design. PAMELA is an easy to follow, REPEATABLE design methodology which can be documented with a drawing tool. Keep in mind though, a drawing tool does not provide any traceability, consistency checking or automated PDL generation.

PAMELA 2 is a second generation of PAMELA 1. in which even the acronym has been changed to reflect the extended applications of PAMELA 1. PAMELA 2 now stands for Pictorial Ada Method for Every Large Application and consists of a standardized, semantically rich, graphical notation which can be applied to the entire software lifecycle.

National Aeronautics and
Space Administration

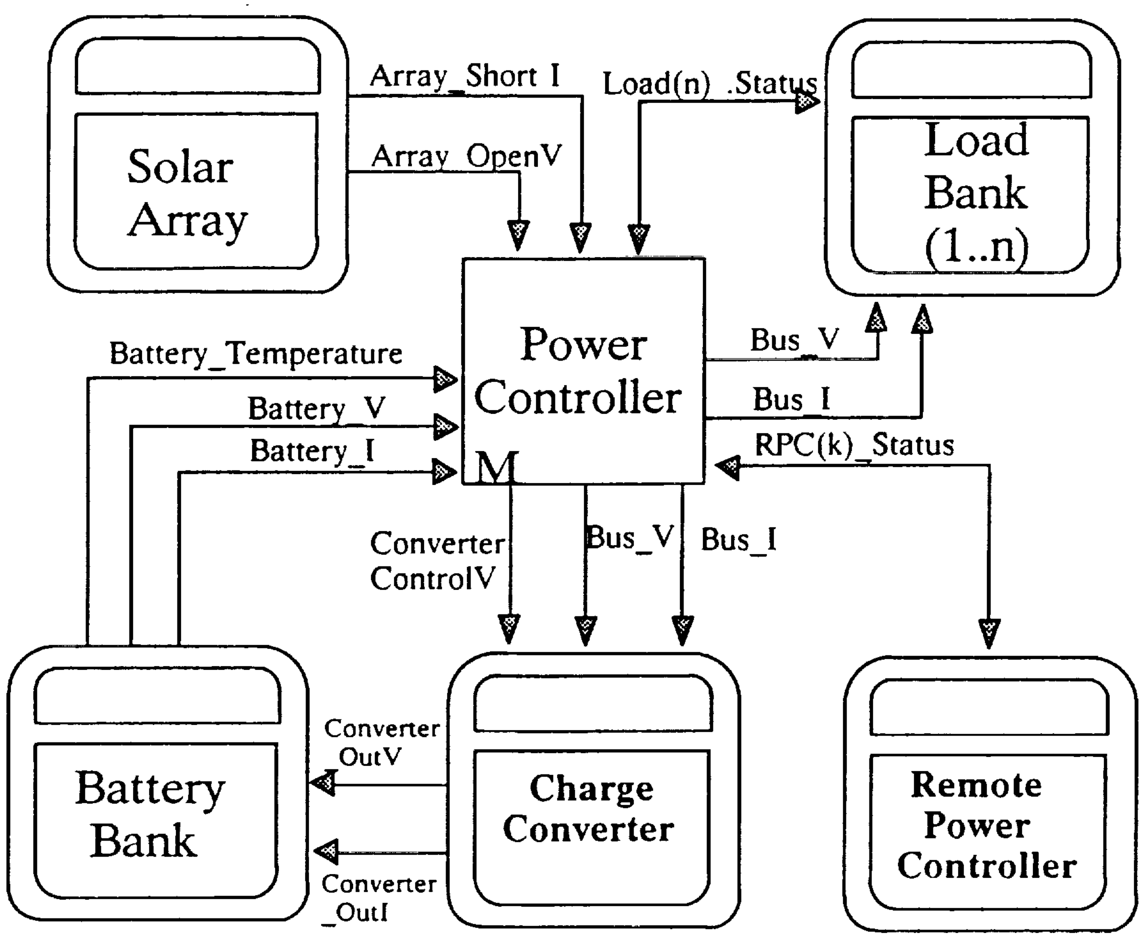
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

External Object Graph



National Aeronautics and Space Administration

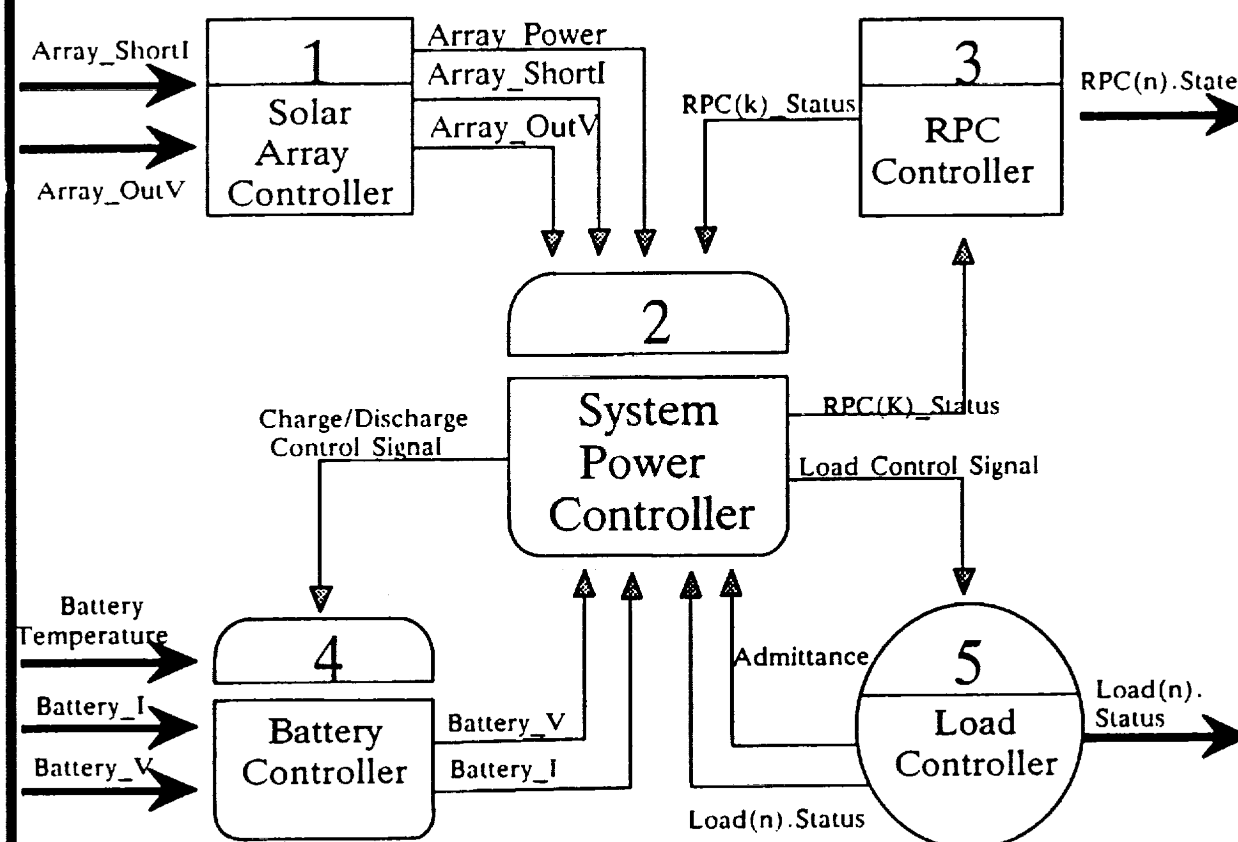
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

System Power Controller Master Subprogram Graph



National Aeronautics and Space Administration

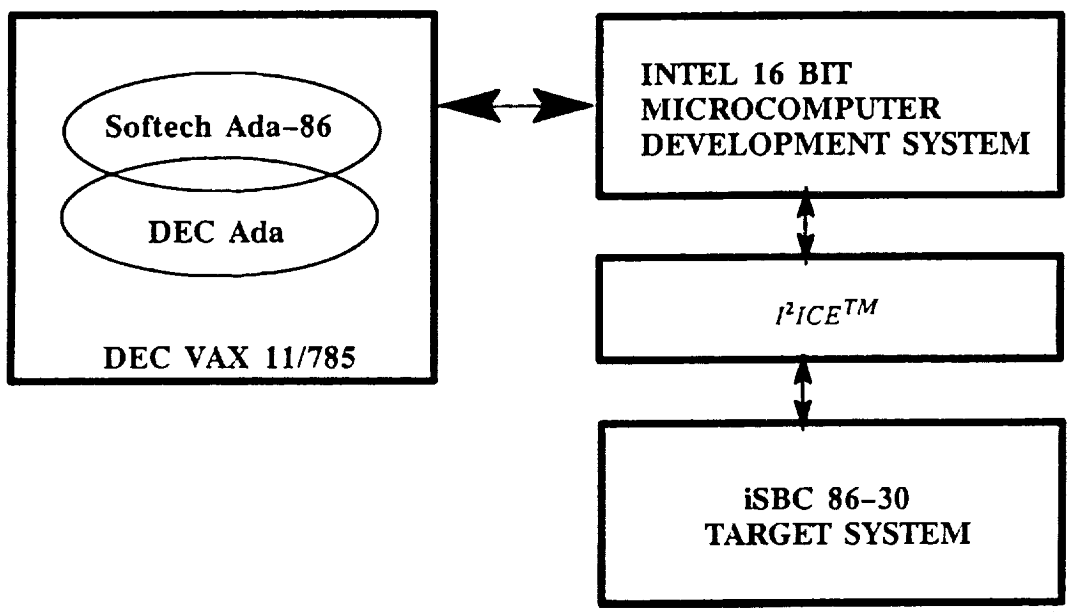
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

Ada Development Environment



The Ada development environment consisted of a DEC VAX 11/785 connected to the INTEL Development System, which was tied to a bare 86-30 single board computer via an in-circuit emulator. This environment proved to be very slow and cumbersome. It became apparent that Ada was not as "transportable" as it claimed to be and that a compiler could pass validation but that did not necessarily mean that it was a production quality compiler. The controls and simulation software could successfully compile and execute on the VAX and complete cross-compilation on the VAX but the execution on the iSBC 86-30 board was beyond the abilities of the Softech Ada-86 run-time environment.

National Aeronautics and Space Administration

Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

Ada vs Fortran

	Ada	Fortran
Lines of Code	1100	1600
Executable Statements	900	1500
# of Modules	13	6
Steady State Execution Speed	No noticeable difference for steady state execution	
Embedded Execution Speed	NOT Real-Time	Real-time

Listed are some empirical relationships drawn from the Ada and Fortran control programs. Even though the metrics are based on the implementation of only one problem, they provide significant evidence supporting the desirability of a high order language such as Ada. The difference in the executable statements is the most notable. This is accounted for by comparing the language constructs in Ada to those in Fortran. For example, exception handling in Ada eliminates the need for flag variables that are repetitively set and checked for fault conditions. Also, the number of modules in Ada is more than double the ones used for the Fortran equivalent. The higher modularity of the Ada program is a direct result of software engineering principles such as a structured design methodology, reusability, and increased efficiency. The steady state execution speed was compared on the VAX 11/785 with no noticeable difference, but, once the application is embedded on the 8086, the execution is bogged down by the run time environment. At this time though, proof of concept was more critical than real-time execution. Also, note that the extent of the listed differences is likely to vary from one application to another and the metrics used are generalizations and should not be used as absolute conclusive results.

Lessons Learned

- Ada requires well-educated software engineers
- **DO NOT** code Ada from another language
- The requirements specification and design will determine the success or failure of a project.
- All Ada compilers are not created equal
  *Both functional and performance differences

The main lesson learned was that Ada requires well-educated software engineers. The training program currently followed includes a week long Introduction to Ada, with hands-on training as a course requirement. This is followed with a course in a software design methodology such as PAMELA or Object Oriented Design. Then, once the development team gains some experience in writing Ada code, a follow up Advanced Ada course is scheduled. A Software Engineering course is also recommended, which includes a discussion of the software lifecycle, its phases, products and activities. Classroom training which provides hands-on experience is the most effective for people ready to start coding in Ada, but for managers a day of Ada terminology and its benefits is more appropriate. Other forms of training such as video tapes or computer aided instruction are available to anyone at any time.

The objectives of this project were to demonstrate and evaluate the abilities and limitations of the Ada programming language for an embedded microprocessor application. Since that time, there has been a vast improvement in the availability and performance of target compilers. The objectives were met and the development team learned a great deal about Ada.

PMAD System Testbed

- Multi–Processor Power System Testbed
- Currently in the design phase, implementing Object Oriented Design techniques
- Configuration managed with the SSE Automated Product Control Environment (APCE)

The PMAD System Testbed is a multi–processor system used to control the hardware as shown in the following diagram. The purpose of the power system testbed software is to provide an environment for testing various control algorithms and newly developed hardware. The software can be broken down into two types: the system environment software and the algorithms under test. The algorithms under test include any algorithms written to control the power system. The Power Management Controller is connected to the other control processors via the Ethernet communications protocol. Processor status and power system component information is available to any processor requesting that information. The Power Component controllers are connected directly to the power component via a 1553 interface. The software is currently in its design phase and the development team is implementing Object Oriented Design techniques. The software development lifecycle is configuration managed with the Space Station Software Support Environment (SSE) Automated Product Control Environment (APCE).

National Aeronautics and Space Administration

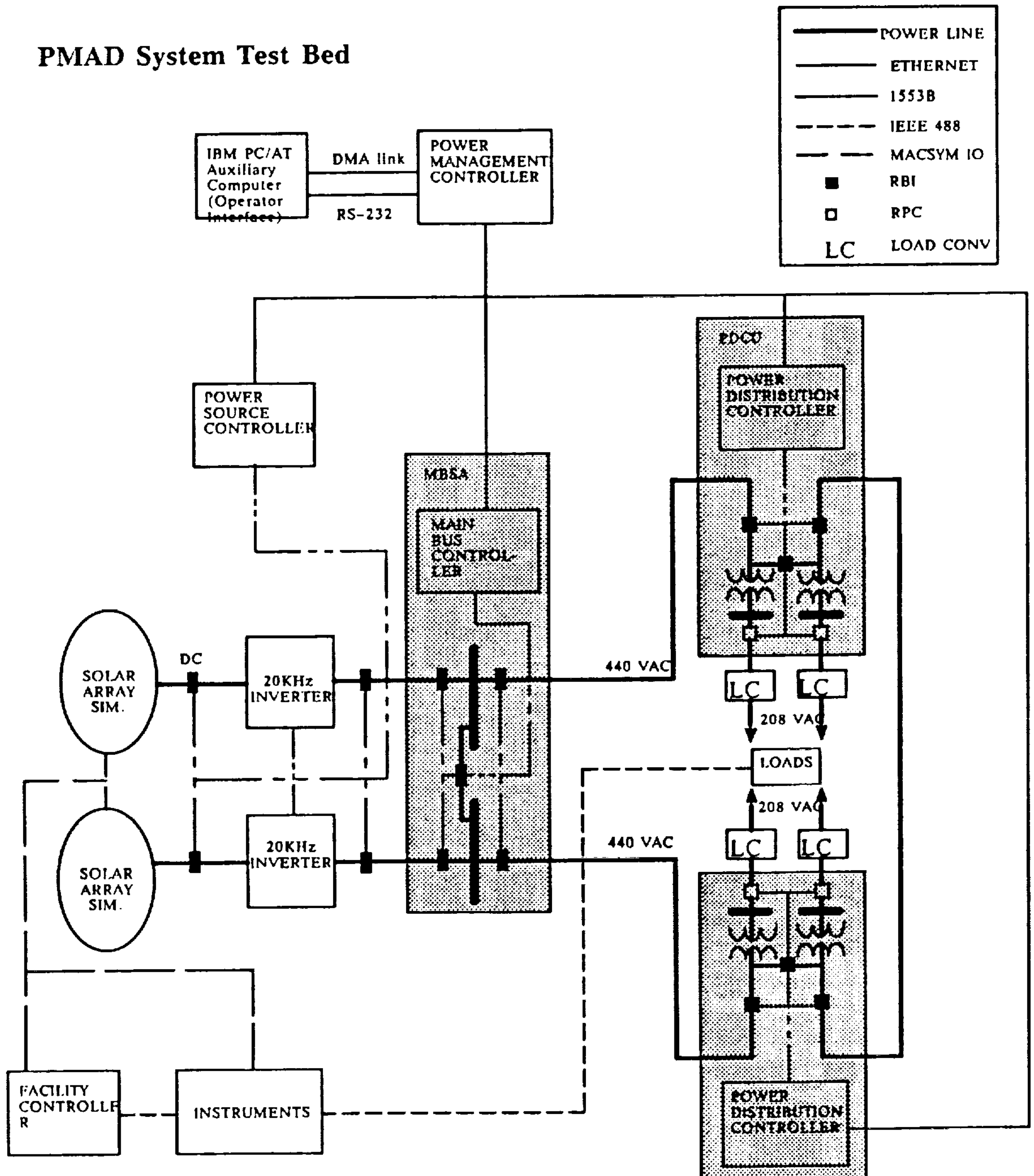
Lewis Research Center

SPACE STATION SYSTEMS

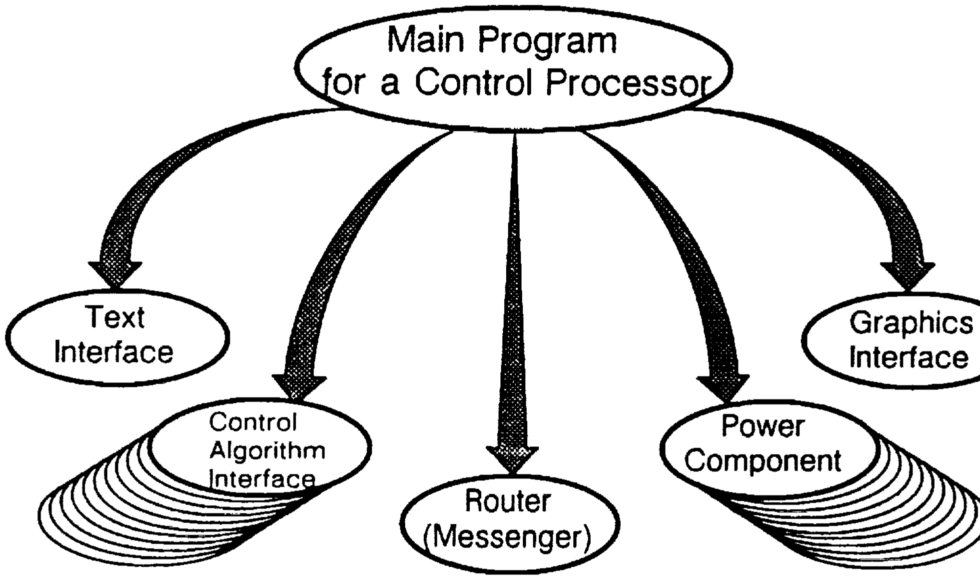
ELECTRICAL SYSTEMS DIVISION

Space Station

PMAD System Test Bed



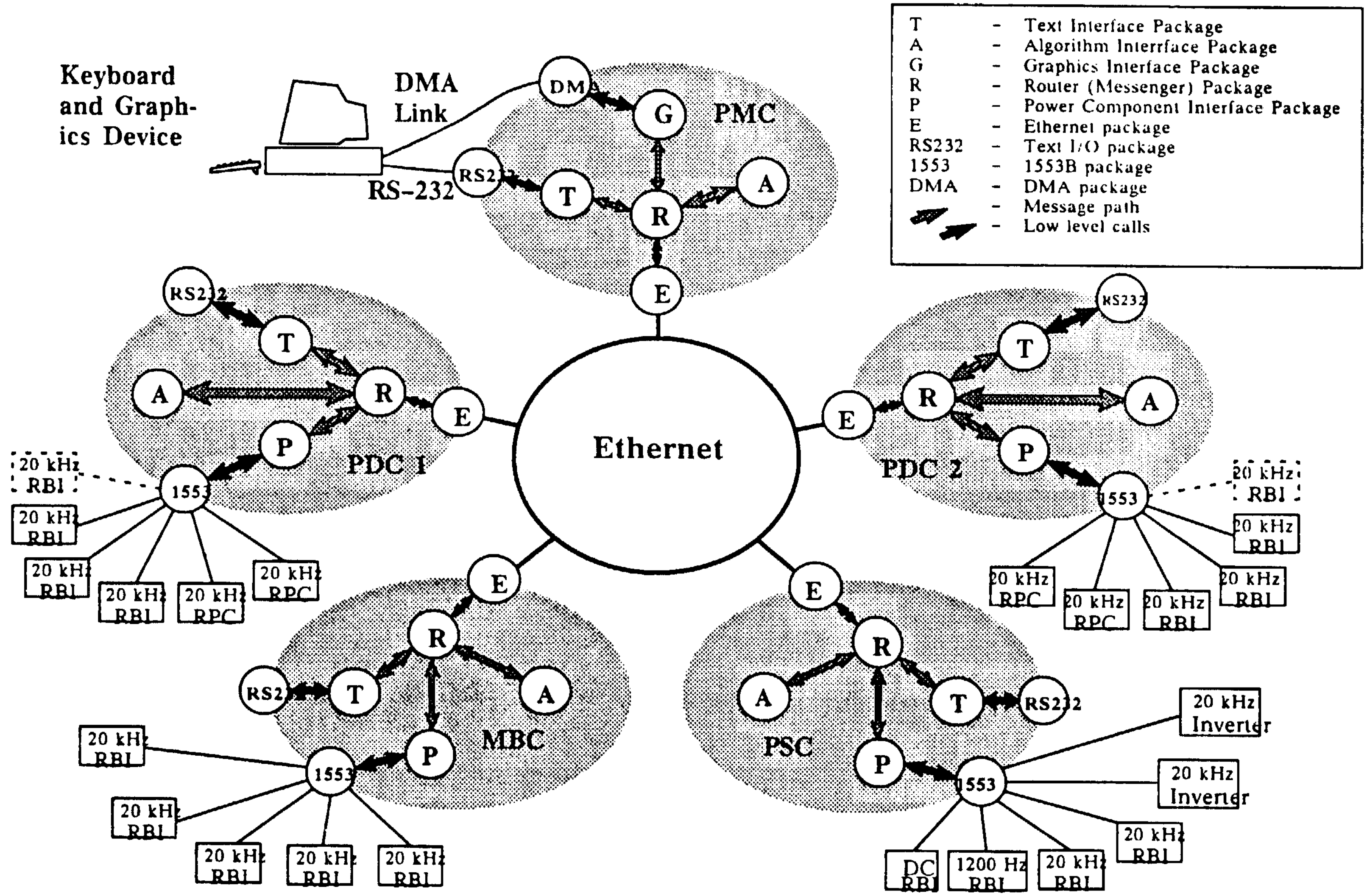
* NOTE all components shown are single string (no redundancy)



System Decomposition Chart for a Single Processor

Each of the distributed processors shall contain the main program, unique to that control processor function, which communicates with a common set of interface packages. These packages include the following: a text interface which provides an operator interface to the system for debugging capabilities; a standard control algorithm interface so that prototype control algorithms may be easily incorporated into the system and tested; a router or messenger package which standardizes all the inter-process communications to the Ethernet; a power component package which communicates to the power components via the 1553 data bus; and a graphics interface which shall receive, interpret and display commands from the PC/AT graphics connection. A functional block diagram is shown in the following diagram. The development team is currently evaluating the ALSYS Ada 8086 family of cross-compilers for this application.

Functional Block Diagram (Package Level)



Experience with the APCE

- The project documentation is under configuration control.
- Traceability pointers have been defined for all the software requirements.
- The mechanics of using the APCE are difficult to learn.
- Currently unable to transfer design diagrams to the SSE mainframe.
- PRC support has been excellent.

The APCE database for the Power System Test Bed Software contains all the documentation under configuration control. The system requirements have been identified, and pointers have been defined which establish the traceability of requirements throughout the lifecycle. The mechanics involved in entering the information into the APCE has proved to be difficult at times. To use the APCE effectively requires that the user learn the APCE project language. For example, the phases, products, and sections are identified with two or three letters, i.e. "RD SR ALL" is the Software Requirements Document, in the requirements definition phase, and includes ALL the sections. Once the project base has been established, the APCE is relatively easy to use for the developers and testers. The tester takes on a major role throughout the software lifecycle by defining test procedures to verify and validate each step in the lifecycle. The PMAD project is currently in the detailed design phase, but at this time we are unable to place the design diagrams under APCE control. Although, as the development team completes their detailed design the APCE team is defining test procedures to run against the code as soon as it is promoted to the APCE. Planning Research Corp., PRC, has provided excellent assistance and guidance throughout the project, particularly in the area of software testing.

National Aeronautics and Space Administration

Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

PMAD Integrated Testbed

- Representative of the Space Station PMAD System.
- Currently in the initiation/requirements definition phase.
- Shall be used to evaluate overall PMAD system performance and to address system level issues.

The PMAD Integrated Testbed (ITB) is a 20kHz power system testbed consisting of the components shown in the following diagram. The major items of the ITB include the DC Switching Units, the Main Bus Switching Units, the Power Distribution and Control Units, and the Main Inverter Units. The software control system shall monitor, evaluate, and control the ITB performance from the power sources to the loads. In addition, the control system shall monitor and control feeder, bus, and component electrical loads. The ITB is currently in its initiation/requirements definition phase and shall be used to evaluate overall PMAD system performance and to address system level issues.

National Aeronautics and Space Administration

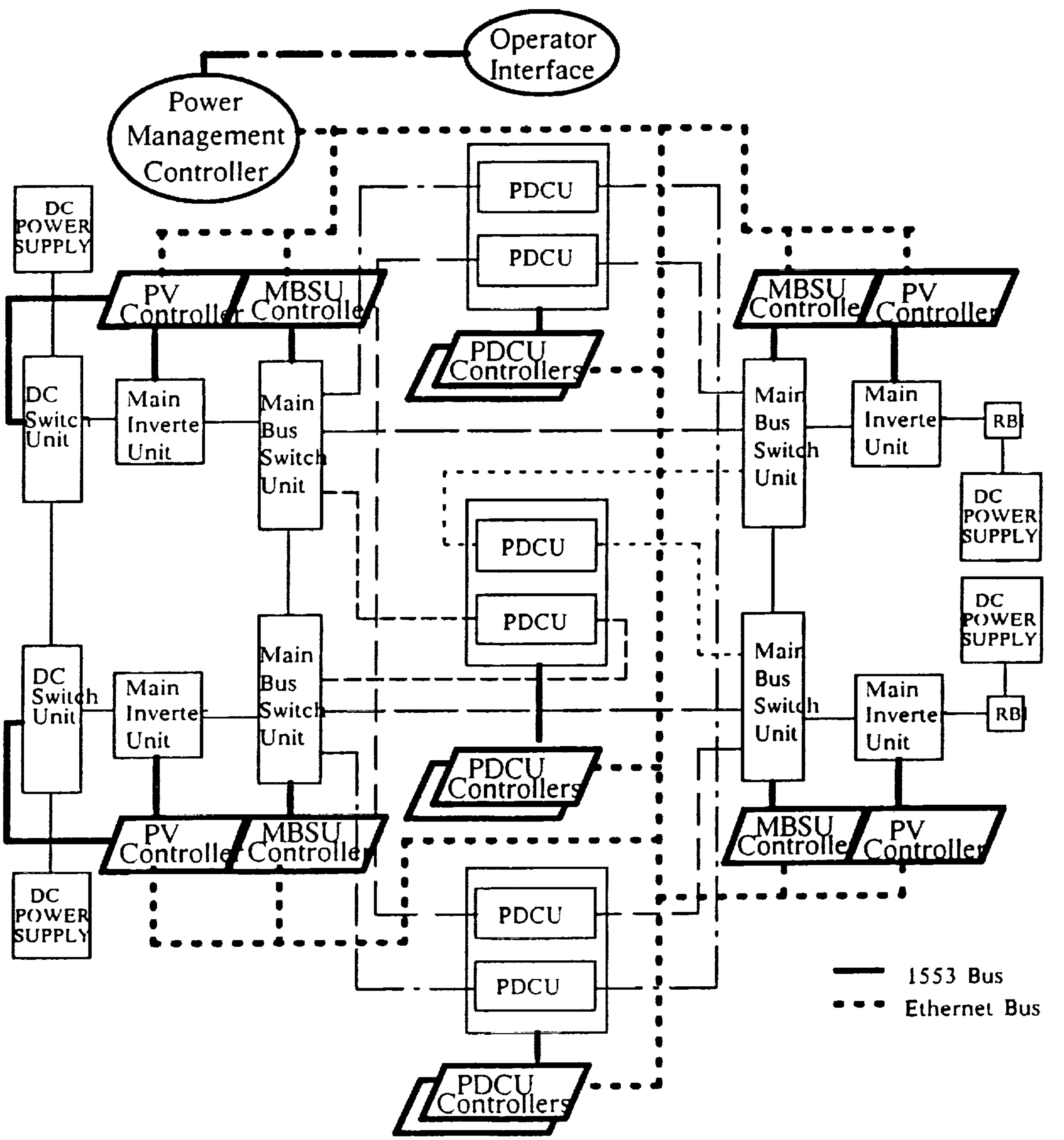
Lewis Research Center

SPACE STATION SYSTEMS

ELECTRICAL SYSTEMS DIVISION

Space Station

PMAD Integrated Test Bed Configuration



Space Station Electrical Power System

- Work Package 04 C/D contractor is Rockwell International, Rocketdyne Division.
- Software Lines of Code Estimation = 90,000 SLOCS
- Software is broken up into 9 CSCIs, the use of Ada is a program requirement.

The Space Station Project is divided into 4 work packages, each divided into two phases. NASA Lewis Research Center and its prime contractor, Rocketdyne, is Work Package 04 and is responsible for the detailed design, development, test, evaluation, and construction of the electrical power system. Initially, power will be provided by eight solar array wings, phase two shall incorporate a solar dynamic power module. The power system software is broken down into nine Computer Software Configuration Items (CSCIs) which include a Power Management Controller, a Node Switching Controller, a Power Distribution Controller, a Main Bus Switching Controller, a Photovoltaic Controller, a Solar Dynamic Controller, a Solar Dynamic Engine Controller, a Main Inverter Unit, and a Frequency Changer Unit. The total estimated software lines of code are 64,800 SLOCS and will be written in Ada. The Space Station Software Support Environment tools, rules and standards shall apply to all operational software for the Space Station.

Conclusion

- Space Station is committed to Ada
- Space Station software demands embedded, real–time performance
- Ada compiler technology must improve

In conclusion, the Space Station project is committed to the use of Ada. NASA Lewis Research Center has been involved in the implementation of Ada for the Power Management and Distribution System for over three years and have confronted major issues in the use of Ada, of which all of these can be overcome with the improvement in Ada host and target compiler technology. The Ada language itself requires intensive training in the use of Ada as well as in modern Software Engineering techniques. Finally, the Space Station imposes very stringent demands on the capabilities of the Ada language and the compiler technology has to keep pace with these demands for the application of Ada to be successful.

FIRST NASA ADA USERS SYMPOSIUM

USING ADA;

AN EARLY SPACE STATION FREEDOM EXPERIENCE

BRANDON L. RIGNEY AND CORA L. CARMODY

12/1/88

USING ADA

CONTENTS

USING ADA

BACKGROUND

THE SSE OI 2.0 TRANSFORMATION PROCEDURES PROJECT

- SOFTWARE TO TRANSFORM TEXT AND GRAPHIC OBJECTS
    - NON-TRIVIAL, FREQUENTLY USED

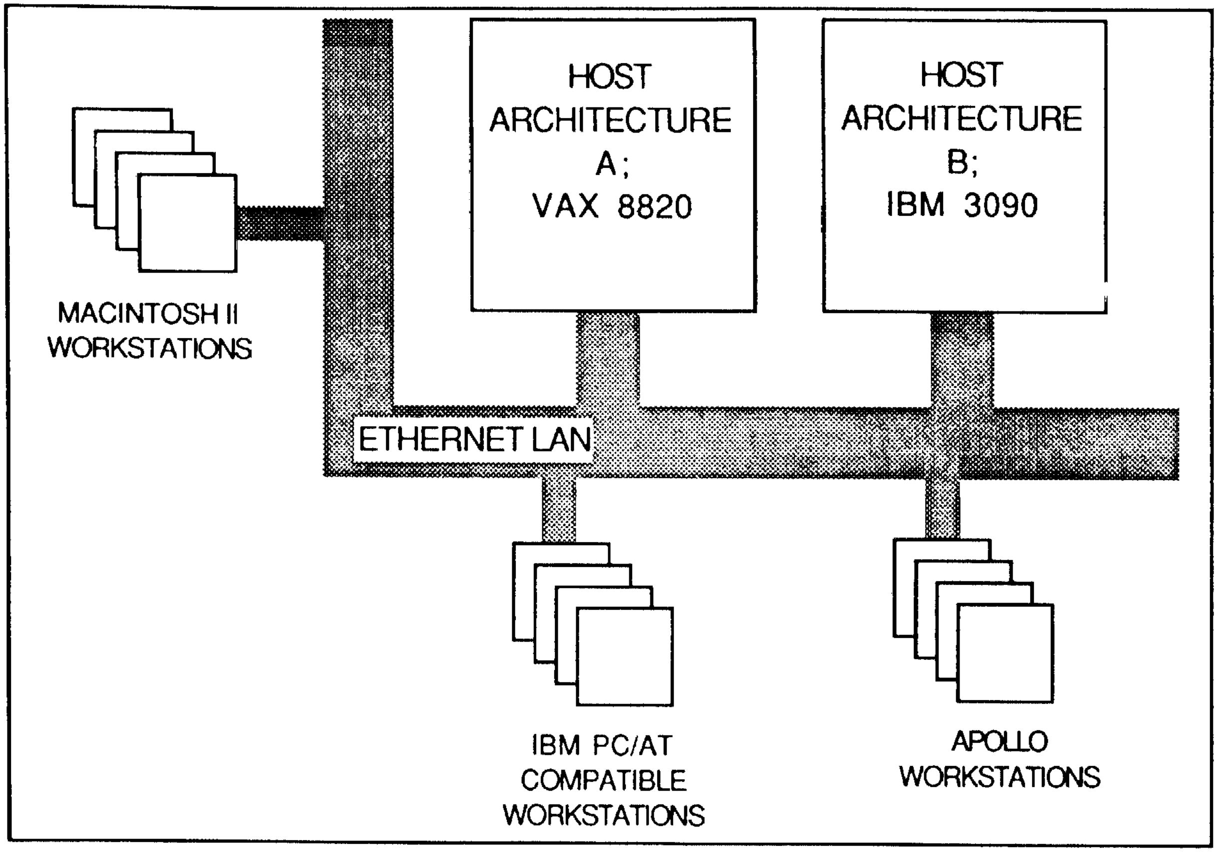
- FIRST PROJECT TO COMPLETE ADA DEVELOPMENT USING THE SSE

- HIGH MOTIVATION TO DESIGN FOR REUSE, DUE TO HETEROGENOUS NATURE OF SSE

- OVERCAME COMPILER MATURITY PROBLEMS TO SUCCESSFULLY REHOST

- 32K OF ADA SLOC EFFECTIVELY BECAME 96K SLOC THROUGH REUSE AND REHOSTING

SSE OVERVIEW



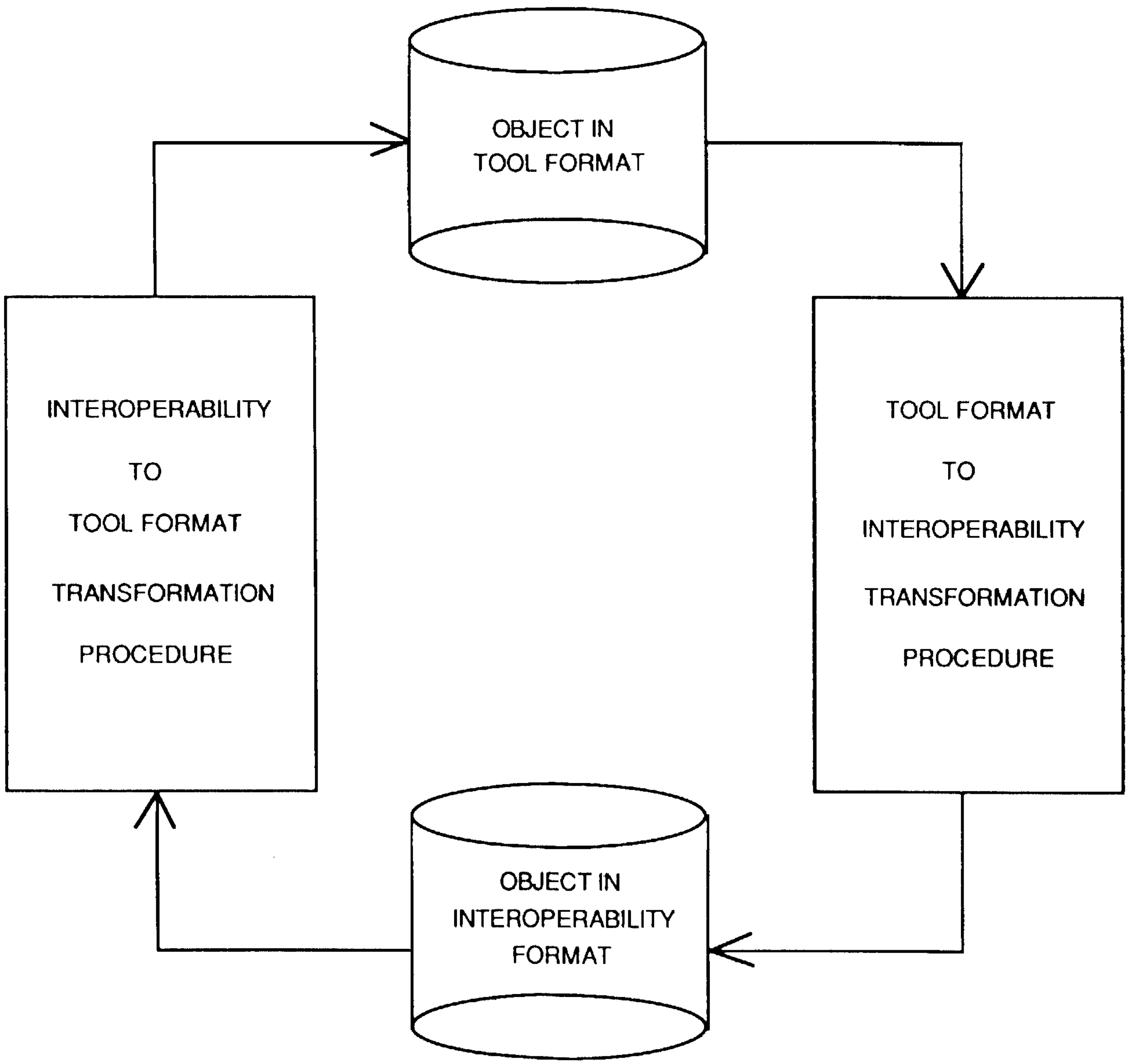
ARCHITECTURE AT THE SSEDF

USING ADA

ASPECTS OF THE TRANSFORMATION PROCEDURE SOFTWARE

- COMPLEX APPLICATION; PARSING DIVERSE GRAMMARS
- MINIMAL, LINE-ORIENTED USER INTERFACE
- MINIMAL, FILE-ORIENTED OPERATING SYSTEM INTERFACE
- EVOLVED FROM PROTOTYPES; REQUIREMENTS NOT PRECISELY STATED PRIOR TO DEVELOPMENT

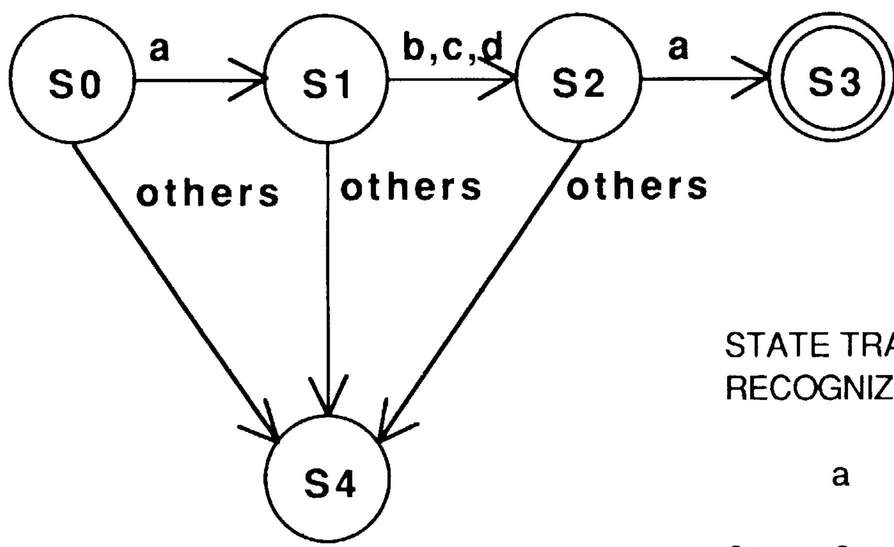
USING ADA



12/1/88

USING ADA

ADA DEVELOPMENT, CONTINUED



STATE TRANSITION MATRIX
RECOGNIZER FOR STRING SET "a{b,c,d}a"

	a	b	c	d
S0	S1	S4	S4	S4
S1	S4	S2	S2	S2
S2	S3	S4	S4	S4
S3	*	*	*	*
S4	*	*	*	*

USING ADA

DIRECT CODING OF TRANSITIONS; USING PASCAL

```
case STATE of
  S0: if token = a then
      begin
        newstate := S1 ;
        action   := add ;
        token    := none ;
      end
    else
      begin
        newstate := S4 ;
        action   := discard ;
        token    := none ;
      end ;
  S1: if token = b or token = c or token = d then
      begin
        newstate := S2 ;
        action   := add ;
        token    := none ;
      end
    else
      begin
        newstate := S4 ;
        action   := discard ;
        token    := none ;
      end ;
  S2: if token = a then
      begin
        newstate := S3 ;
        action   := acceptance ;
        token    := str ;
      end
    else
      begin
        newstate := S4 ;
        action   := discard ;
        token    := none ;
      end ;
  S3, S4: ;
end case ;
```

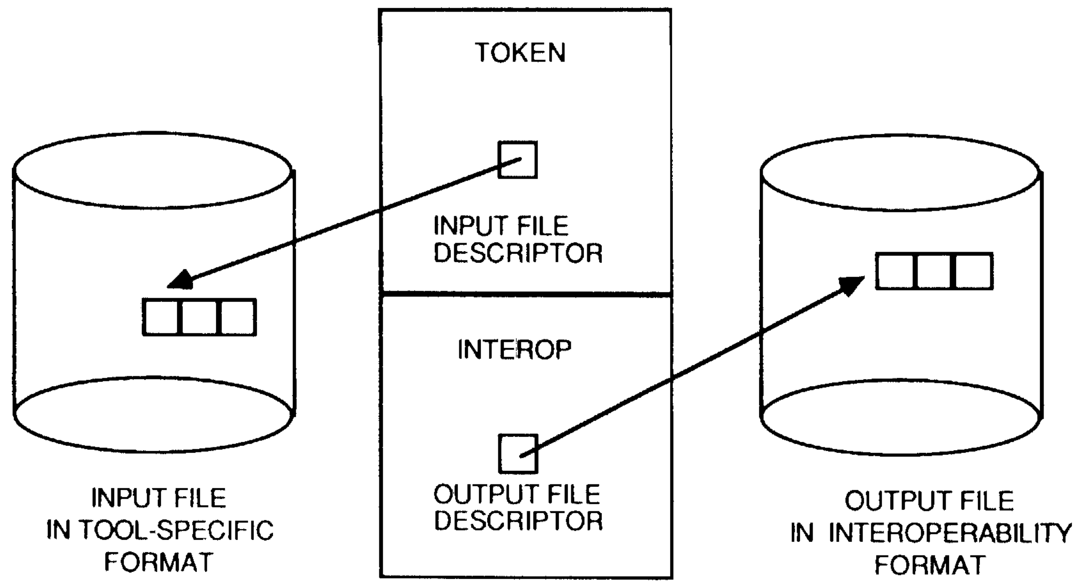
USING ADA

DIRECT CODING OF TRANSITIONS; USING ADA

```
case  state  is
  when S0 => if token = a then
                 next := (S1, add, none) ;
             else
                 next := (S4, discard, none) ;
             end if ;
  when S1 => if token in b..d then
                 next := (S2, add, none) ;
             else
                 next := (S4, discard, none) ;
             end if ;
  when S2 => if token = a then
                 next := (S3, acceptance, str) ;
             else
                 next := (S4, discard, none) ;
             end if ;
  when S3 | S4 => null ;
end case ;
```

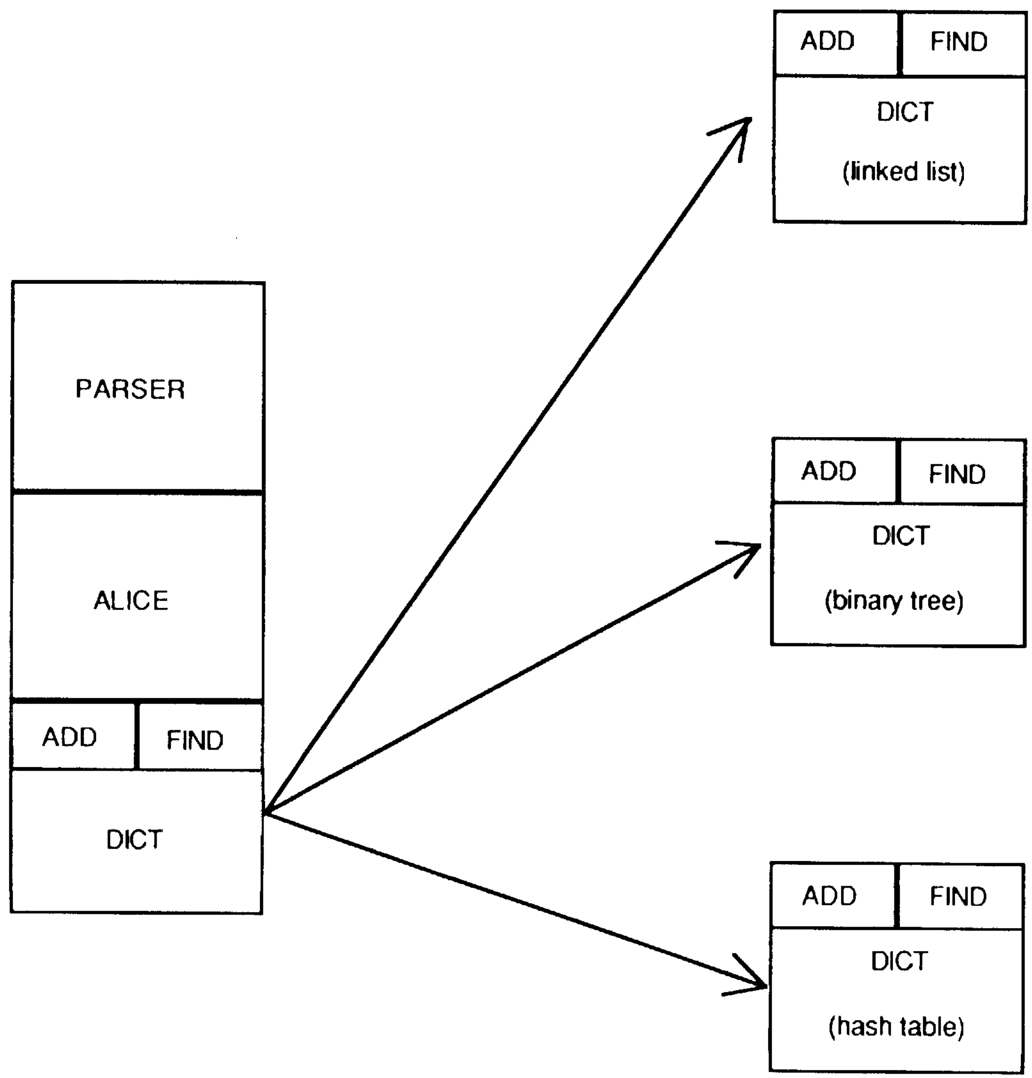
USING ADA

BENEFITS OF PACKAGE DEFINITION



USING ADA

BENEFITS OF PACKAGE DEFINITION, CONTINUED



USING ADA

ADA DEVELOPMENT; OVERLOADING

THE SOLUTION IN OTHER LANGUAGES:

```
look_for_token_1(leftparen) ;
look_for_token_2(endof_ine, endoffile) ;

list := null_list ;
add_to_list(list, leftparen) ;
look_for_token(list) ;

list := null_list ;
add_to_list(list, endofline) ;
add_to_list(list, endoffile) ;
look_for_token(list) ;
```

ADA LETS US STATE IT CLEARLY AND CONCISELY:

```
look_for_token(leftparen) ;
look_for_token(endofline, endoffile) ;
```

USING ADA

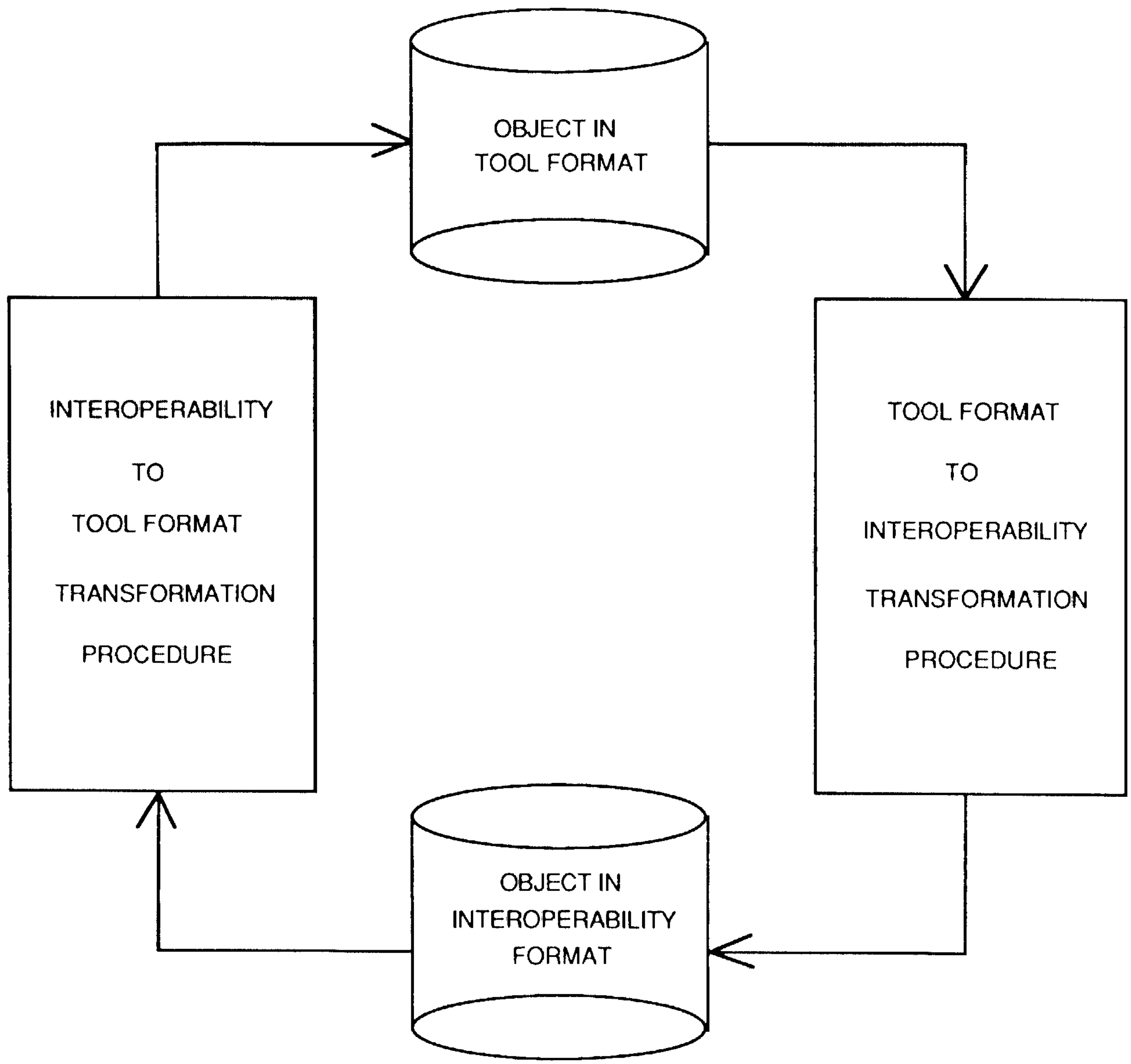
ADA DEVELOPMENT; CLARITY AND MAINTAINABILITY

o AVOID AGGREGATES WHOSE SIZE INTRODUCES CONFUSION. ONE LINE IS A GOOD RULE OF THUMB.

o DECLARING A PACKAGE WILL SAVE LITTLE EFFORT OR CONFUSION IF THERE IS NO DATA STRUCTURE IT CAN HIDE.

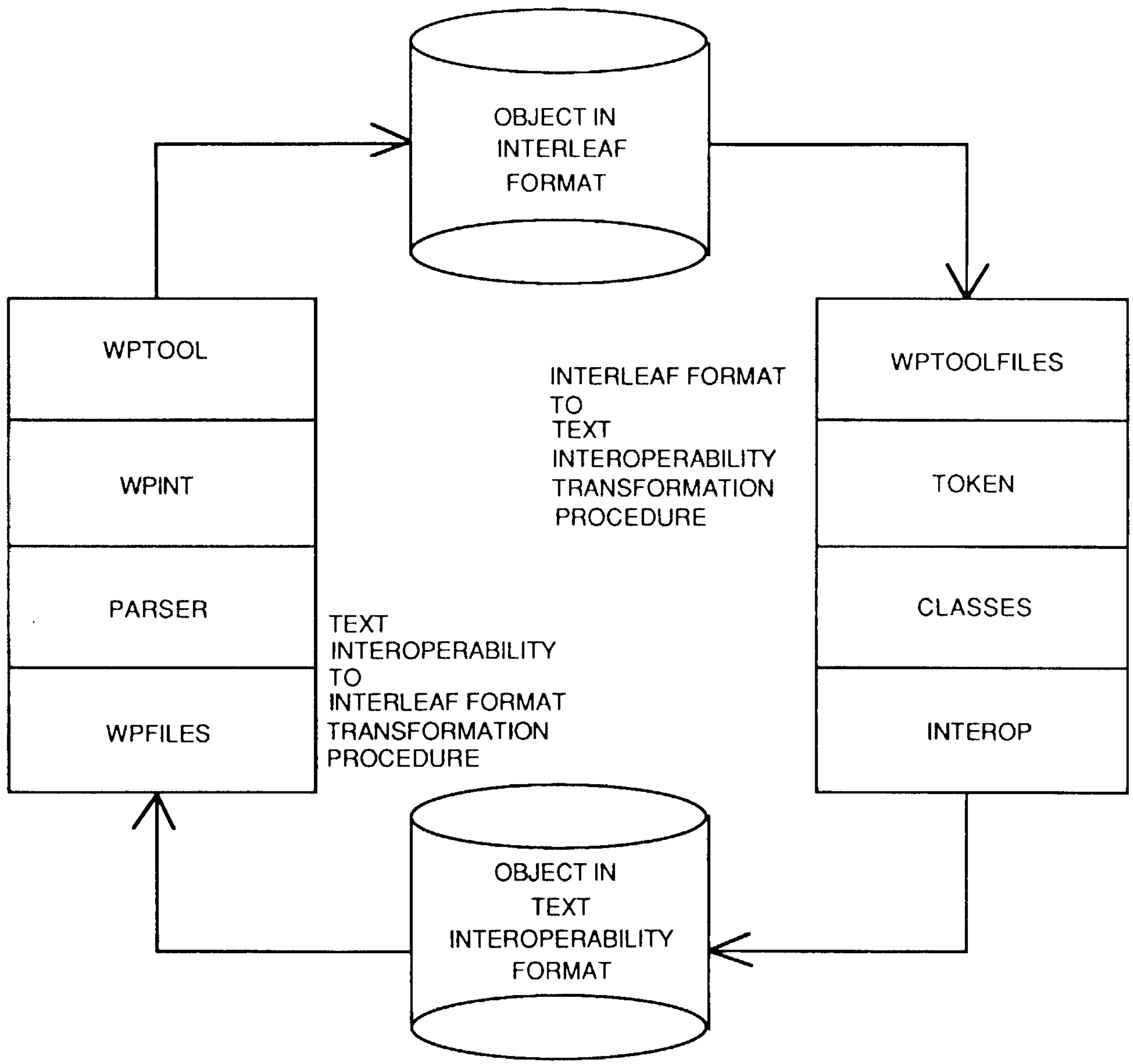
o OVERLOADING AN IDENTIFIER WITH MEANINGS THAT HAVE LITTLE IN COMMON CAN ONLY BE CONFUSING AND MISLEADING TO PROGRAMMERS WHO MIGHT STUDY OR MAINTAIN THE CODE LATER.

USING ADA



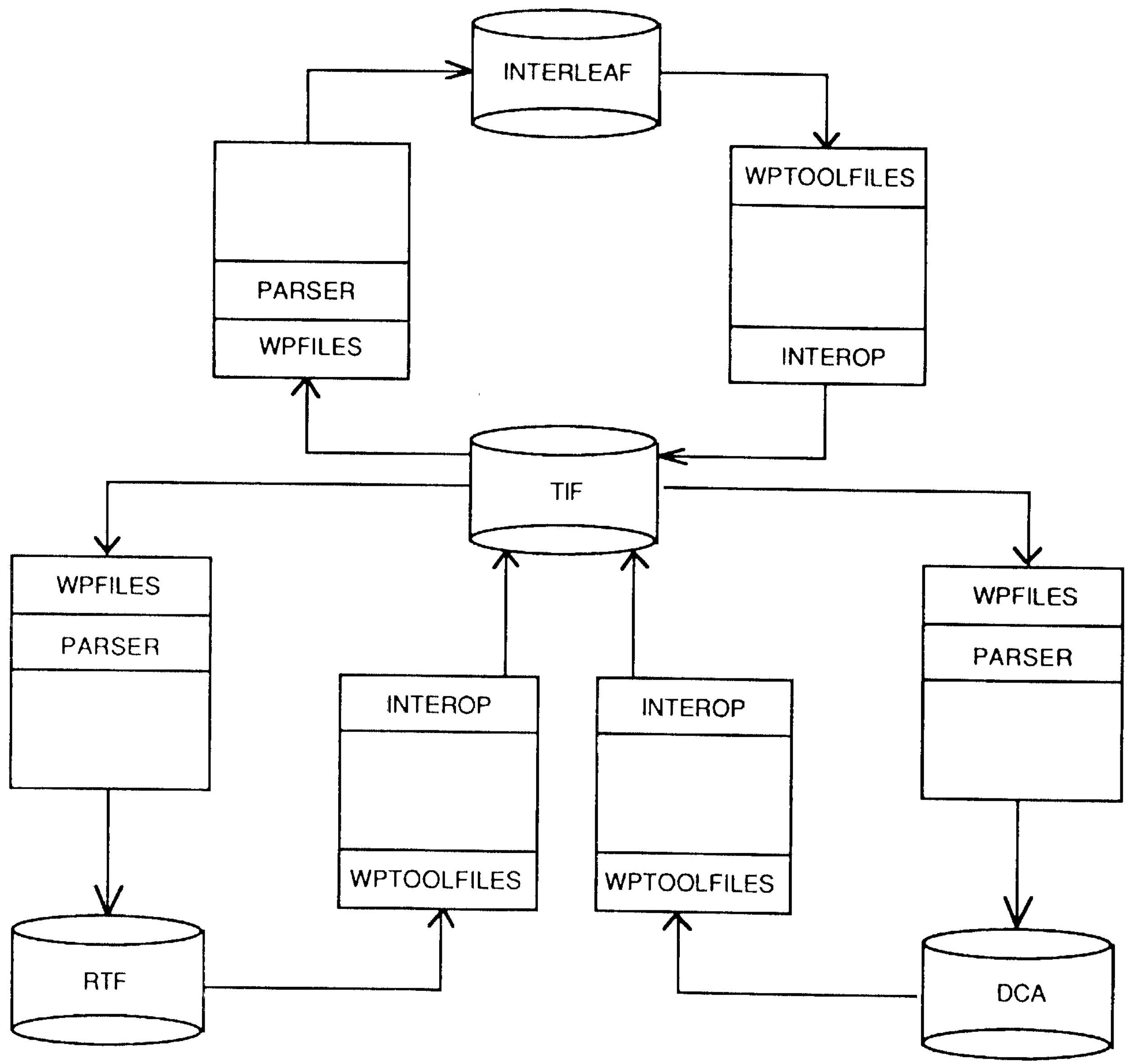
12/1/88

USING ADA



OBJECT IN INTERLEAF FORMAT
WPTOOLFILES
TOKEN
CLASSES
INTEROP
INTERLEAF FORMAT TO TEXT INTEROPERABILITY TRANSFORMATION PROCEDURE
OBJECT IN TEXT INTEROPERABILITY FORMAT
WPTOOL
WPINT
PARSER
WPFILES
TEXT INTEROPERABILITY TO INTERLEAF FORMAT TRANSFORMATION PROCEDURE

USING ADA

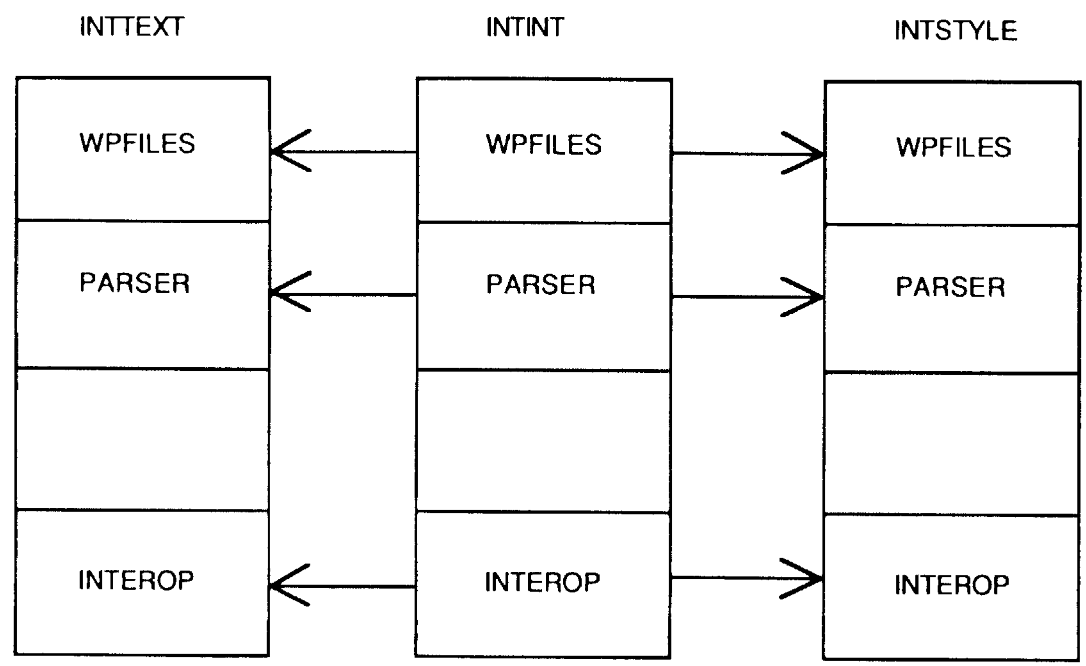


12/1/88

EMHART PRC

USING ADA

REUSE OF DEVELOPED COMPONENTS



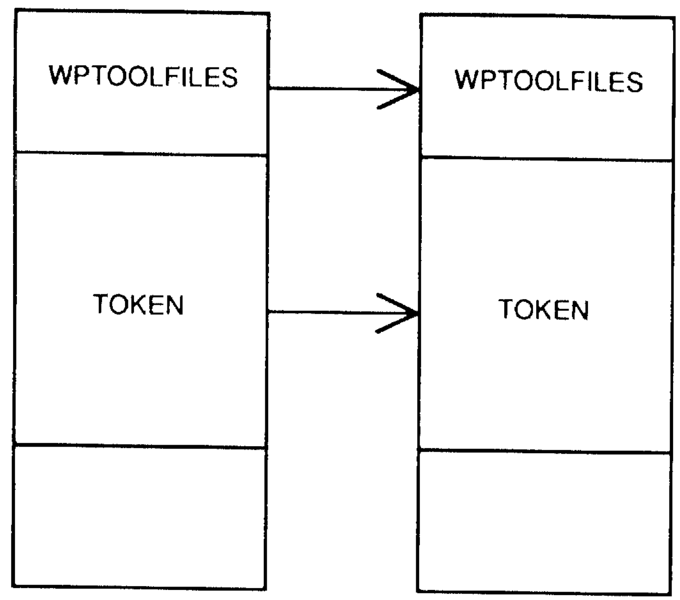
STRIPS INTEROPERABILITY CONTROL STRUCTURES FROM TEXT INTEROPERABILITY FILES

NORMALIZES TEXT INTEROPERABILITY FILES

INFERS AND APPLIES STYLE STRUCTURES TO TEXT INTEROPERABILITY FILES

USING ADA

REUSE OF DEVELOPED COMPONENTS



LEAFINT

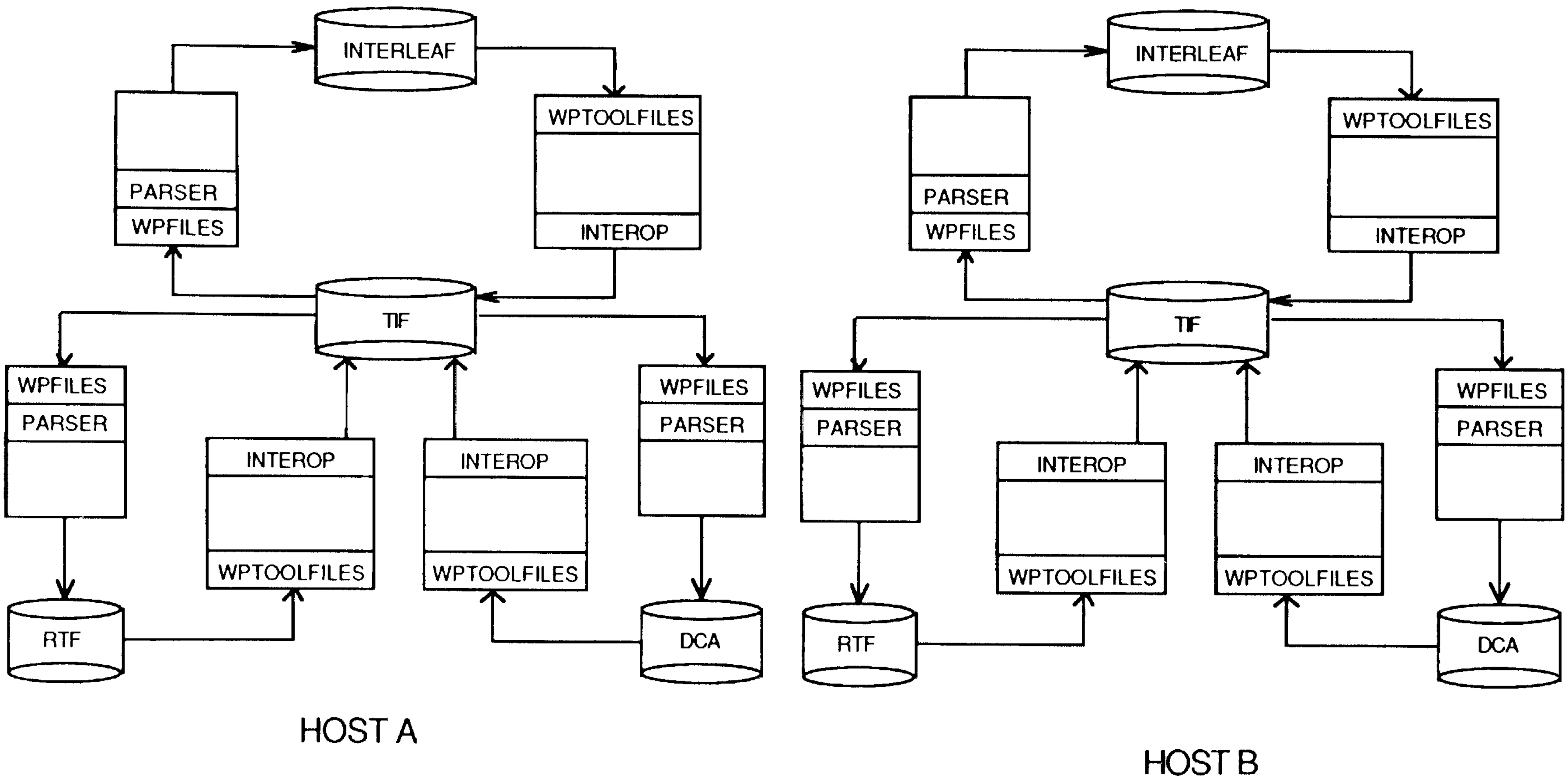
(TEXT)

LEAFINTG

(GRAPHICS)

USING ADA

REHOSTING IS REUSE IN DISGUISE

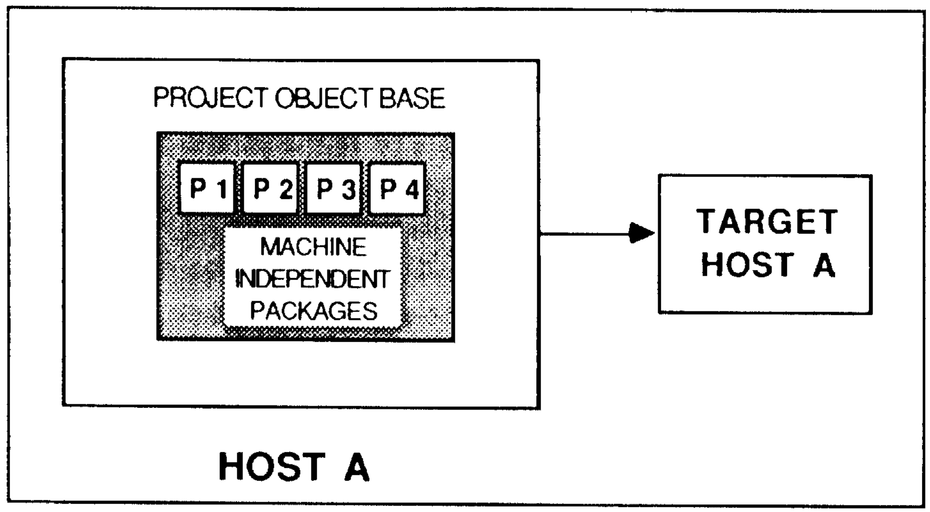


12/1/88

USING ADA

REHOSTING OF ADA, CONTINUED

THE ASSUMPTION;



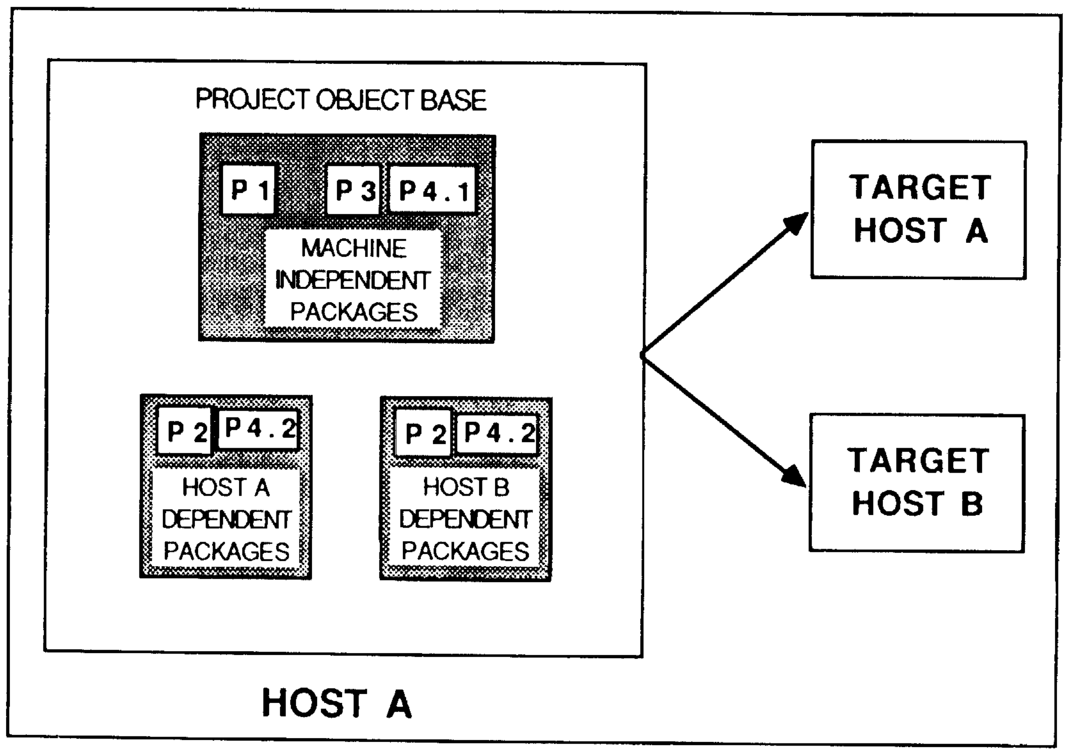
WHEN TARGETING ADA FOR ONE HOST, IT IS EASY TO ASSUME THAT ALL PACKAGES DEVELOPED WILL BE MACHINE INDEPENDENT

AFTER ALL, PORTABILITY WAS ONE OF THE MAIN DRIVERS IN THE DEVELOPMENT OF ADA

USING ADA

REHOSTING OF ADA, CONTINUED

THE REALITY;

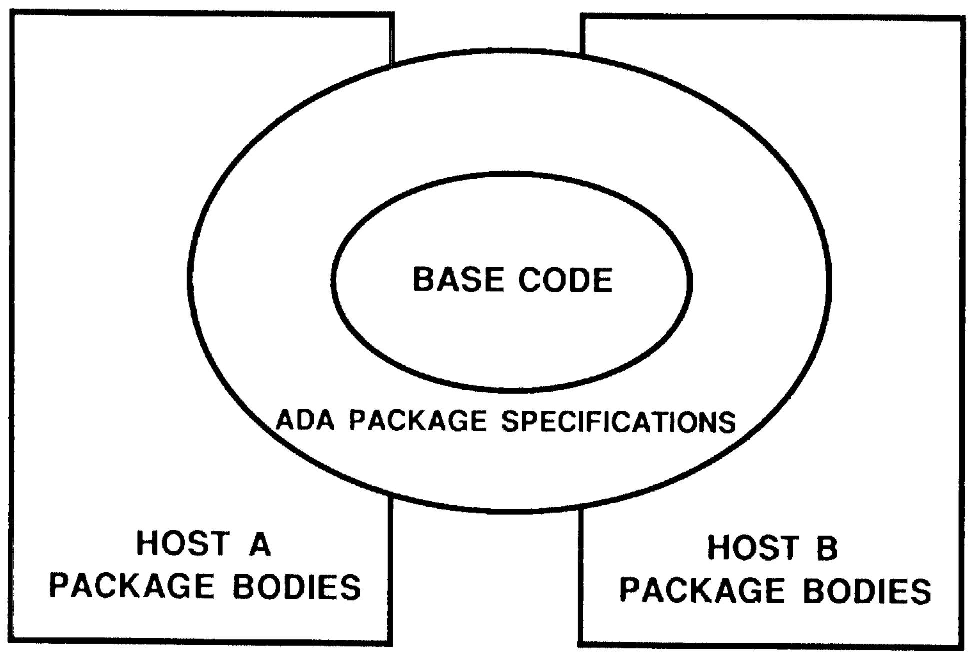


THROUGH THE PROCESS OF REHOSTING, MACHINE DEPENDENCIES ARE DETECTED, REWORKED, AND ISOLATED

USING ADA

REHOSTING OF ADA, CONTINUED

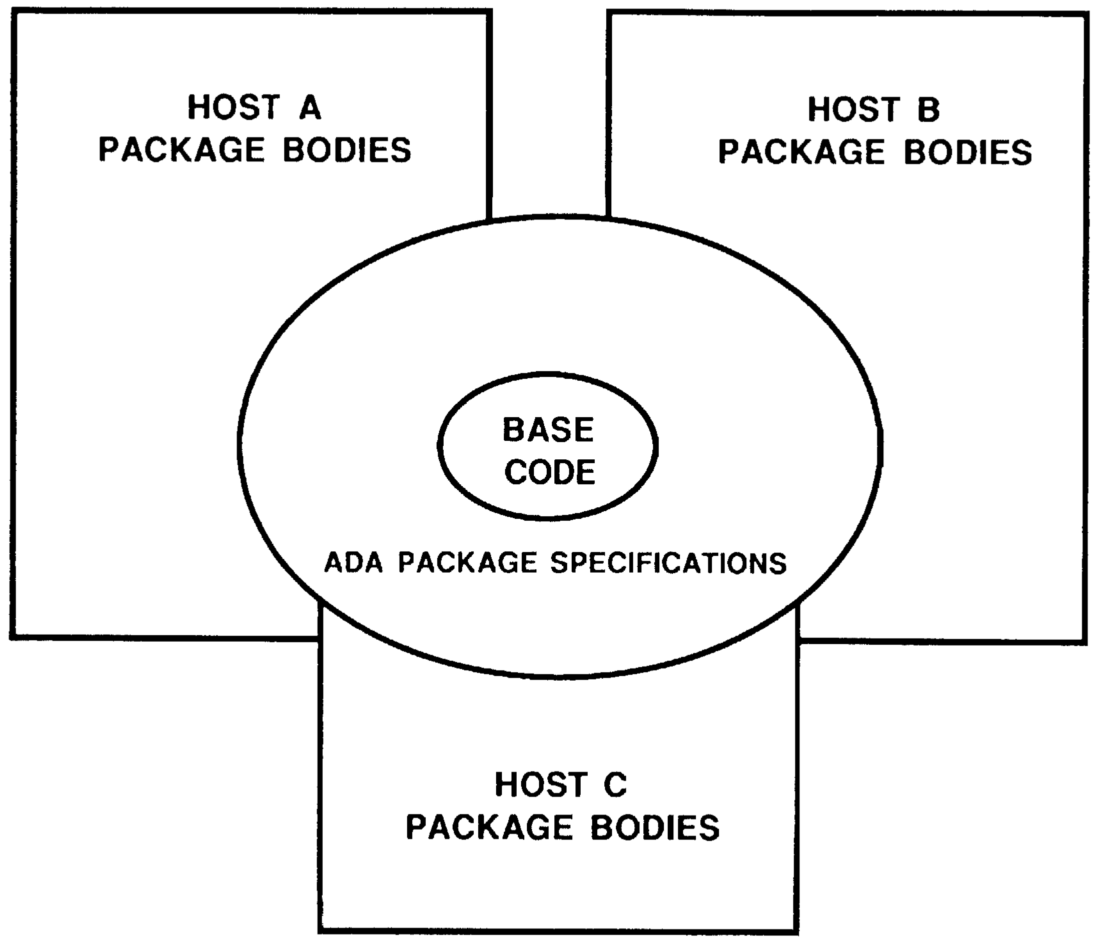
WHAT PORTABLE SOFTWARE LOOKS LIKE;



USING ADA

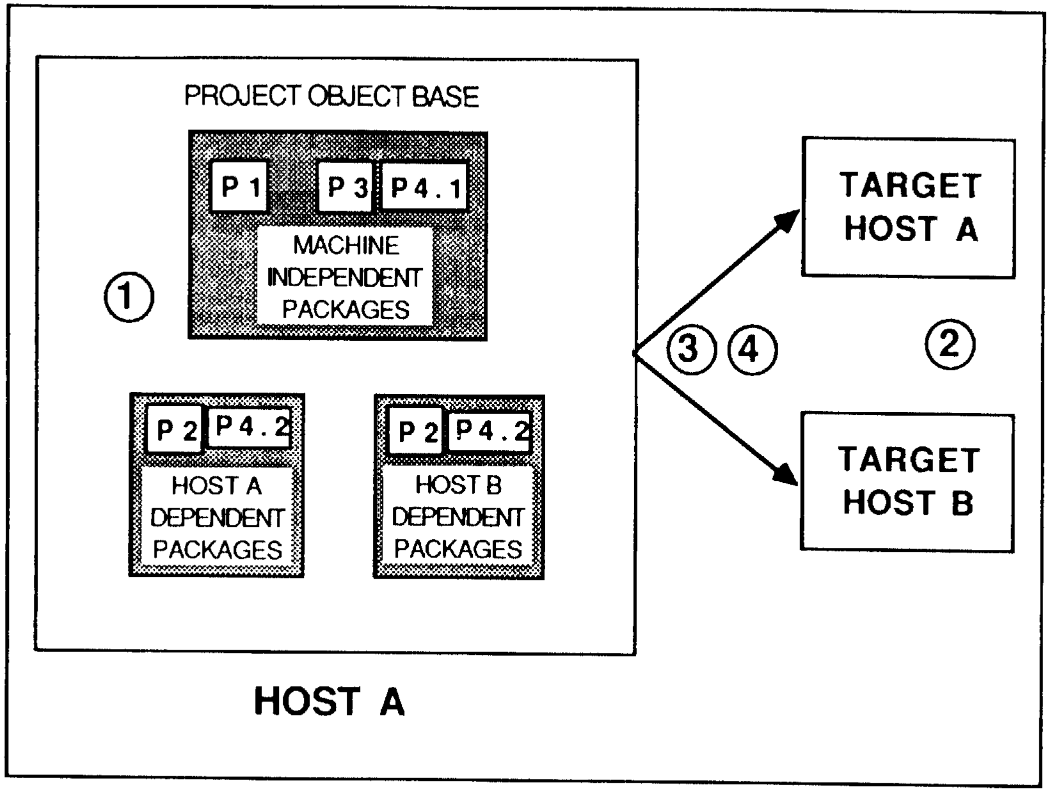
REHOSTING OF ADA, CONTINUED

IF YOU ADD A HOST; MACHINE INDEPENDENT CODE GETS POTENTIALLY SMALLER



USING ADA

SOME POINTS WHERE SPECIAL ATTENTION IS REQUIRED



1) DESIGNING AND IMPLEMENTING THE SOFTWARE TO REDUCE MACHINE DEPENDENCIES
2) COMPILER AND RUN TIME ENVIRONMENT DIFFERENCES
3) TARGET TESTING
4) CONTROLLED CONSTRUCTION OF EXECUTABLES

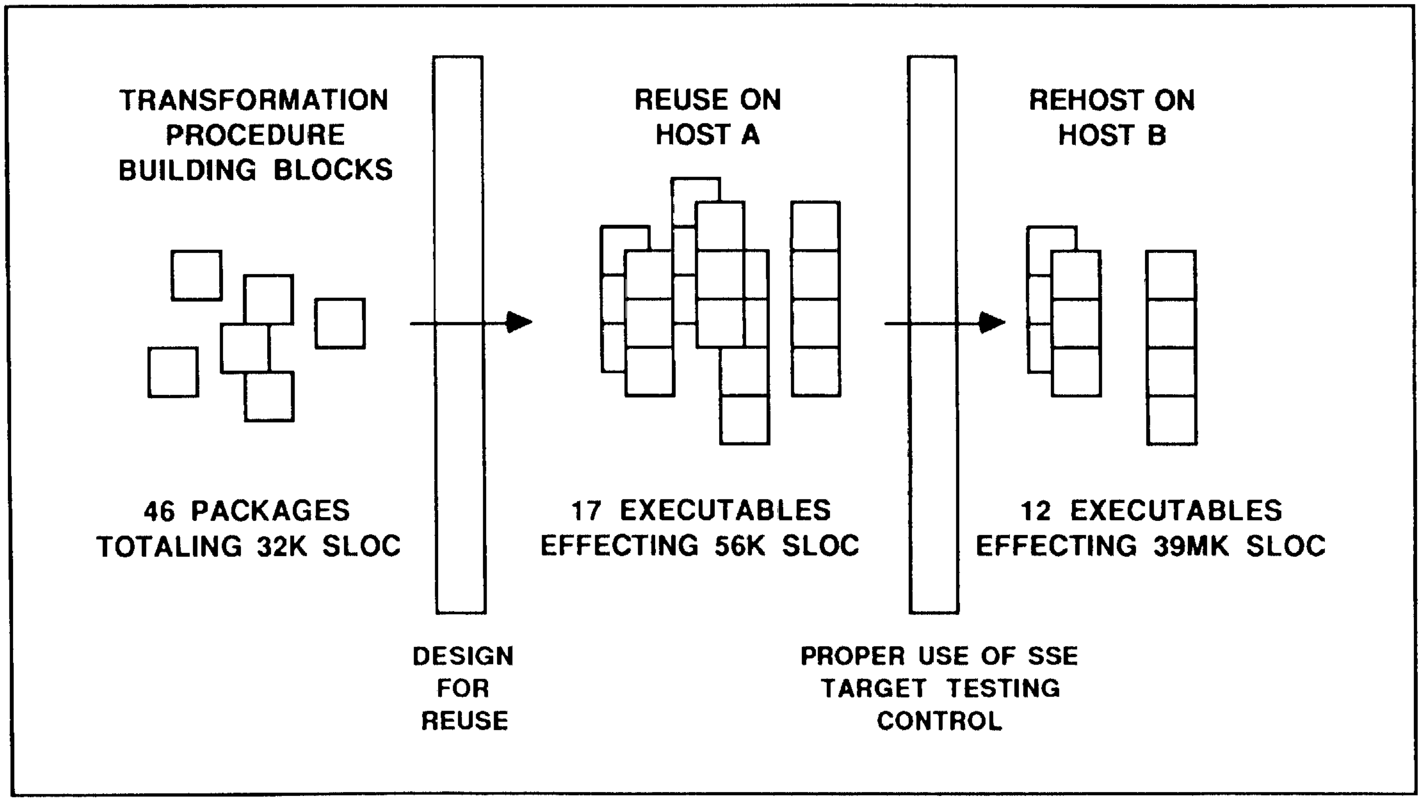
USING ADA

MANAGEMENT OF ADA DEVELOPMENT & TEST

- o ROBUST, REUSABLE SOFTWARE WAS DEVELOPED
- o PRODUCTIVITY WAS HIGH
- o THE TEXT TRANSFORMATION PROCEDURES
  - 6 WEEKS
  - 16K SLOC OF ADA,
  - 4 PROGRAMMERS AND TESTED BY 4 TESTERS.
  - 65K SLOC A DAY PER PERSON
- o WHEN THE EXECUTABLES ARE BUILT
  - 42K EFFECTIVE LINES OF CODE (REUSE AND REHOSTING)
  - 175 EFFECTIVE LINES OF CODE A DAY

USING ADA

MANAGEMENT OF ADA DEVELOPMENT & TEST, CONTINUED



FROM 32K LINES OF CODE, OBTAINED 56.5K + 39.5K = 96K EFFECTIVE LINES OF CODE

USING ADA

SUMMARY

<u>ADA WORKS!</u>

THE USE OF ADA CAN PRODUCE ROBUST, REUSABLE SOFTWARE

WITH MARKED PRODUCTIVITY IMPROVEMENTS

GIVEN;

- o STRONG FRAMEWORK FOR DEVELOPMENT, TEST, AND MAINTENANCE
- o DESIGN GOAL OF REUSE

SSE SYSTEM PROJECT

Ada HOSTS, WORKSTATIONS, AND CROSS-COMPILER

EVALUATION REPORT

D. L. B(Ada)L

12/1/88

AGENDA

PURPOSE:

APPROACH:

HOST/WORKSTATION COMPILER EVALUATION:

CROSS-COMPILER EVALUATION:

CONCLUDING REMARKS:

SSE SYSTEM PROJECT

PURPOSE:

TO PERFORM A TECHNICAL EVALUATION OF Ada HOST/WORKSTATION AND CROSS-COMPILERS FOR USE ON THE SSE. RESULTS OF THIS TECHNICAL EVALUATION TO BE PRESENTED AT PDR.

SSE SYSTEM PROJECT

APPROACH:

- SURVEY OF ADA COMPILERS TO BE EVALUATED
- SELECTION OF CRITERIA TO BE APPLIED
- PIWG BENCHMARKS FOR PERFORMANCE EVALUATION
- COMPILATION TIME ANALYSIS

AGENDA

PURPOSE:

APPROACH:

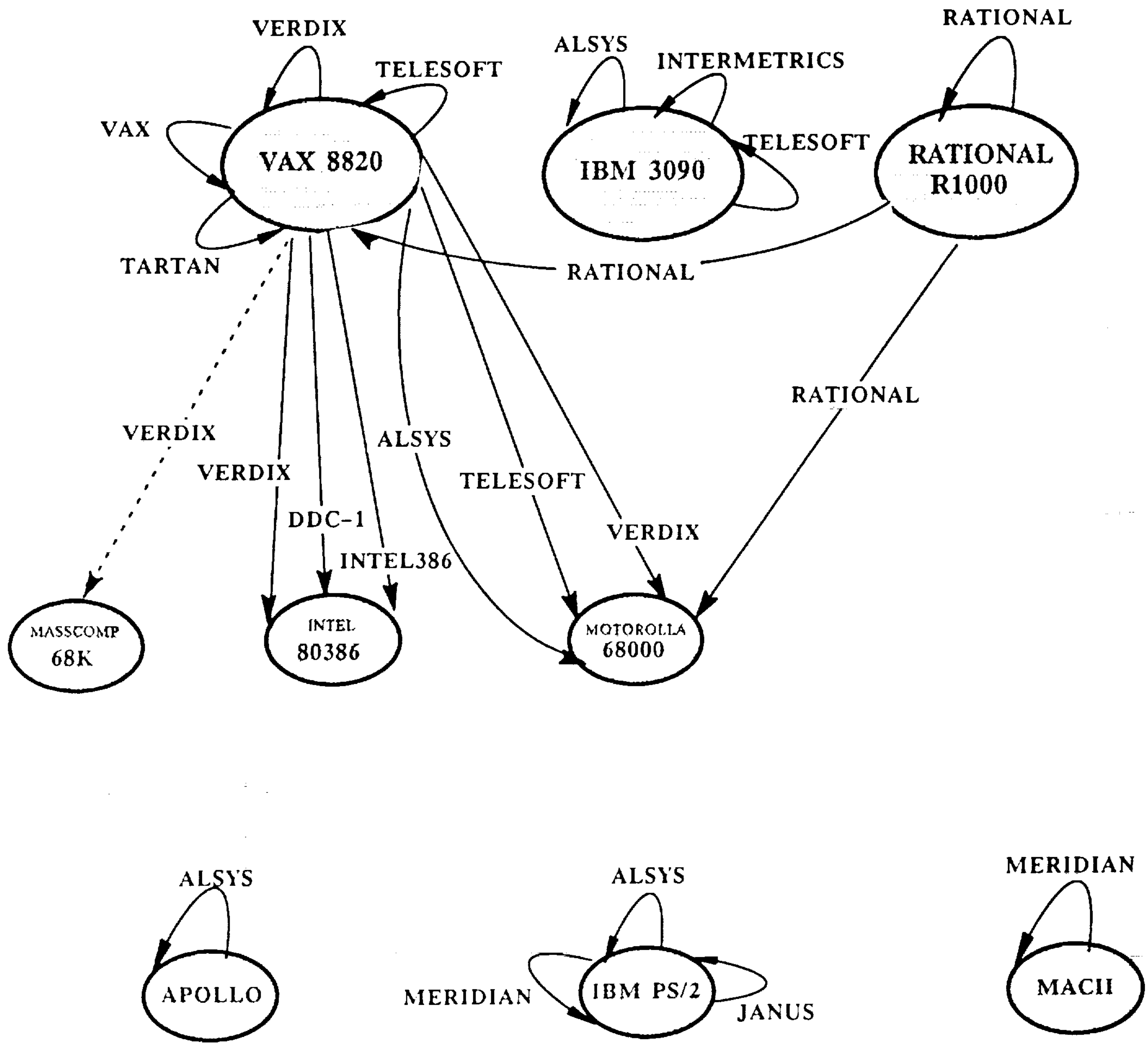
* HOST/WORKSTATION COMPILER EVALUATION:

CROSS-COMPILER EVALUATION:

CONCLUDING REMARKS:

SSE SYSTEM PROJECT

ADA BASE AND CROSS-COMPILER EVALUATION REPORT



SSE
LOCKHEED

ADA COMPILER SSE EVALUATION CATEGORY DESCRIPTIONS

COMPILATION FEATURES

* COMPILE & LINK
* ADA LIBRARY
* DEBUGGER
* ADA SENSITIVE EDITOR

PRAGMAS

* PREDEFINED

CHAPTER 13

* REPRESENTATION CLAUSES
* UNCHECKED PROGRAMMING

TASKING

* SCHEDULING
* IO BLOCKING
* DEADLOCK

DOCUMENTATION

* LRM
* USERS GUIDE
* RUNTIME
* ONLINE HELP

PIWG BENCHMARKS

* PIWG COMPOSITE
* TRACKER
* TASK CREATION
* ARRAY ELABORATION
* EXCEPTION
* CHAPTER 13
* PROCEDURE
* TASKING
* DELAY
* COMPILATION SPEED
* DISK SPACE RQMTS.

RUNTIME

* GARBAGE COLLECTION
* SYSTEM SERVICES
* EXCEPTIONS

MATURITY

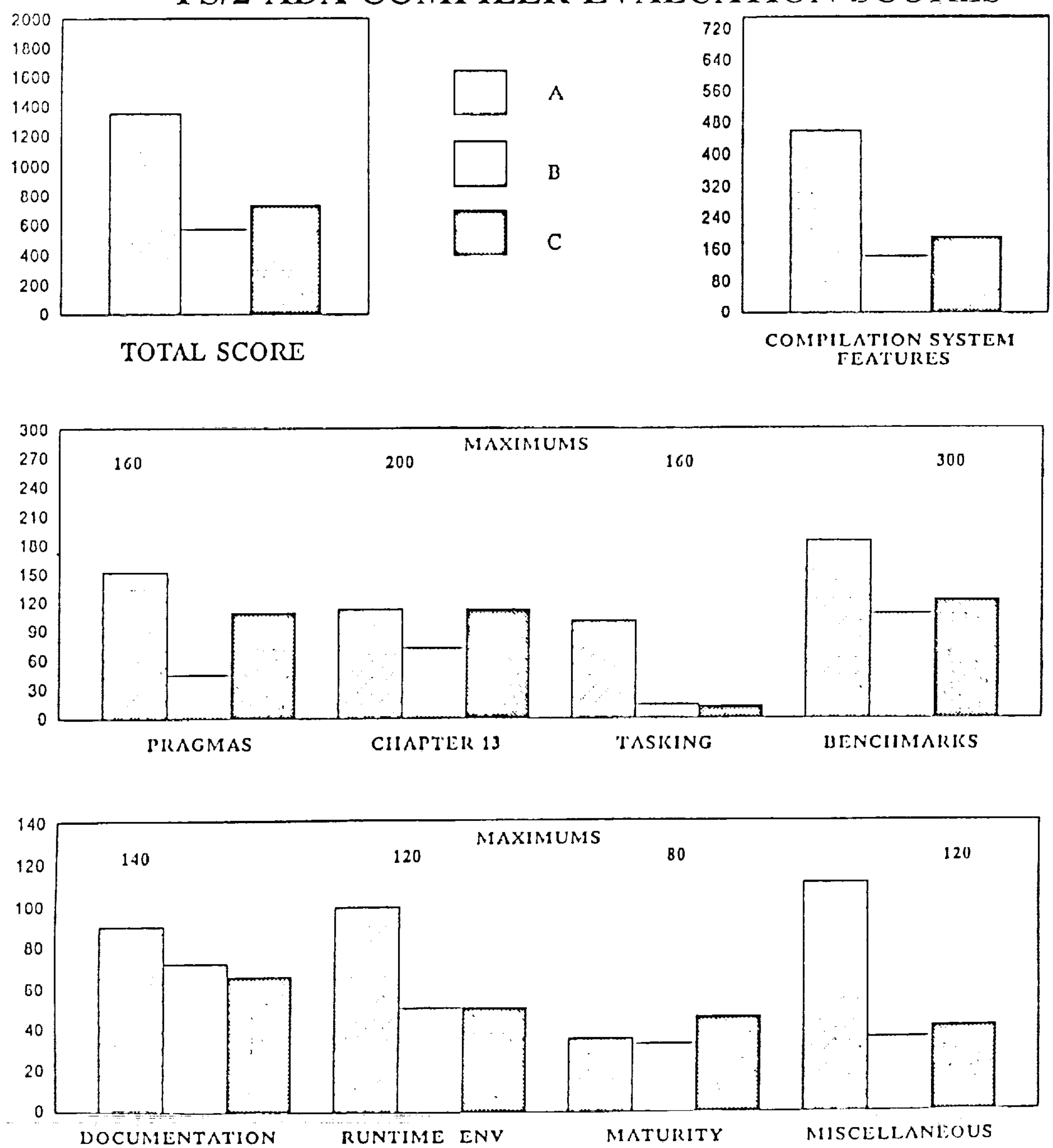
* AGE
* ROBUSTNESS
* OPERATIONAL CONSTRAINTS

MISCELLANEOUS

* SELF COMPILED ADA
* UNIQUE FEATURES

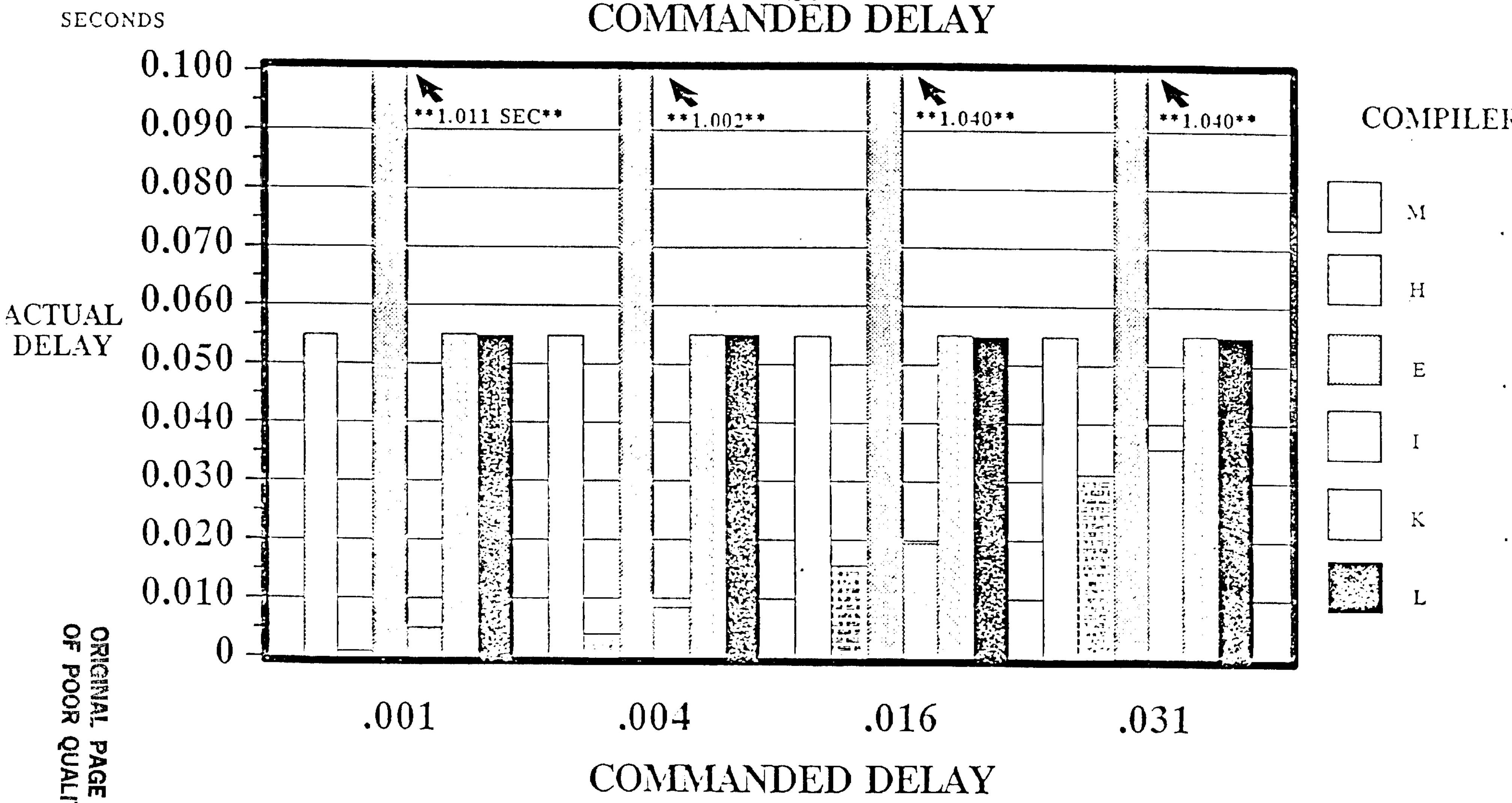
LOCKHEED

PS/2 ADA COMPILER EVALUATION SCORES



LOCKHEED

ACTUAL DELAY VS. COMMANDED DELAY



ORIGINAL PAGE IS OF POOR QUALITY

AGENDA

PURPOSE:

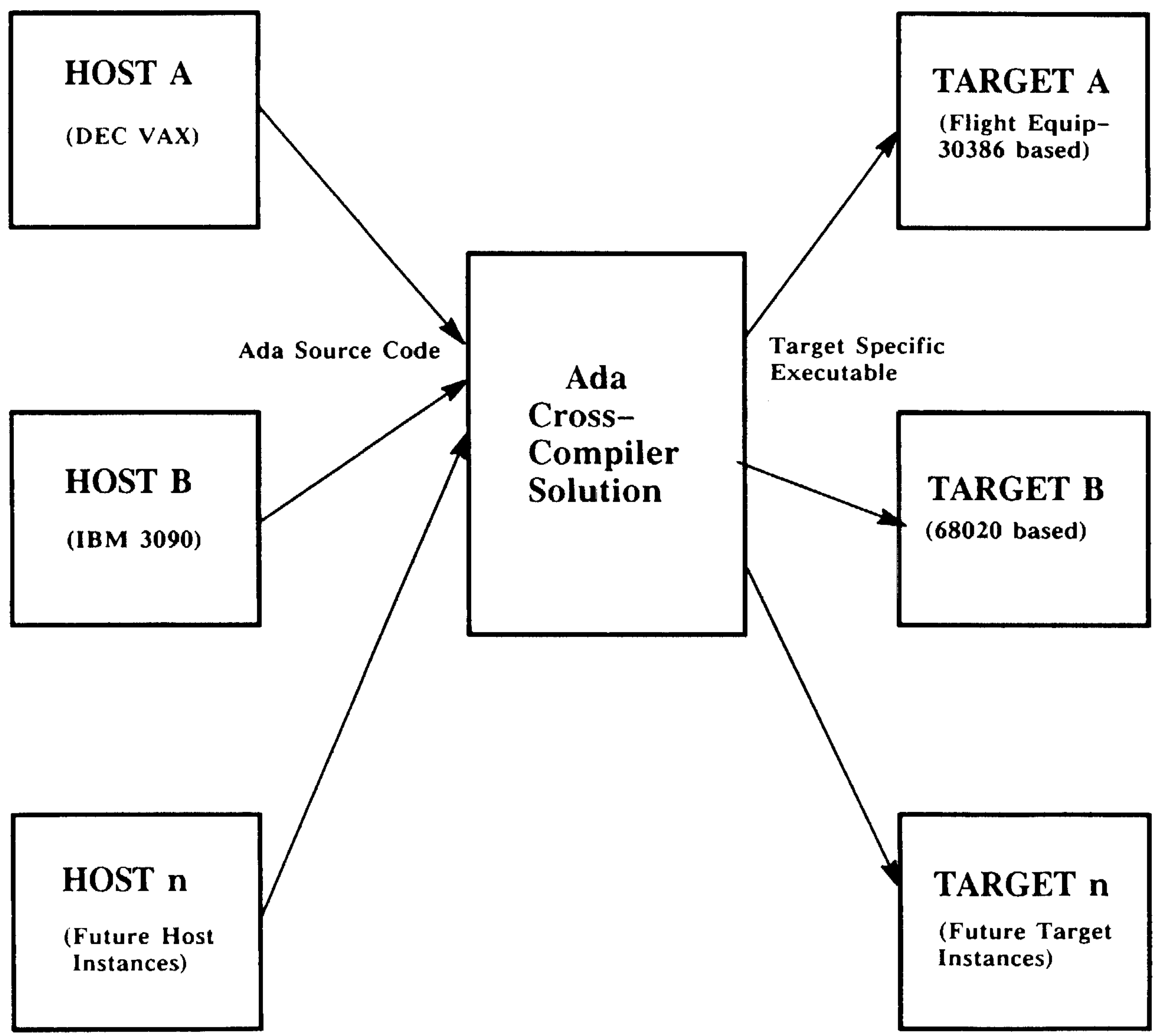
APPROACH:

HOST/WORKSTATION COMPILER EVALUATION:

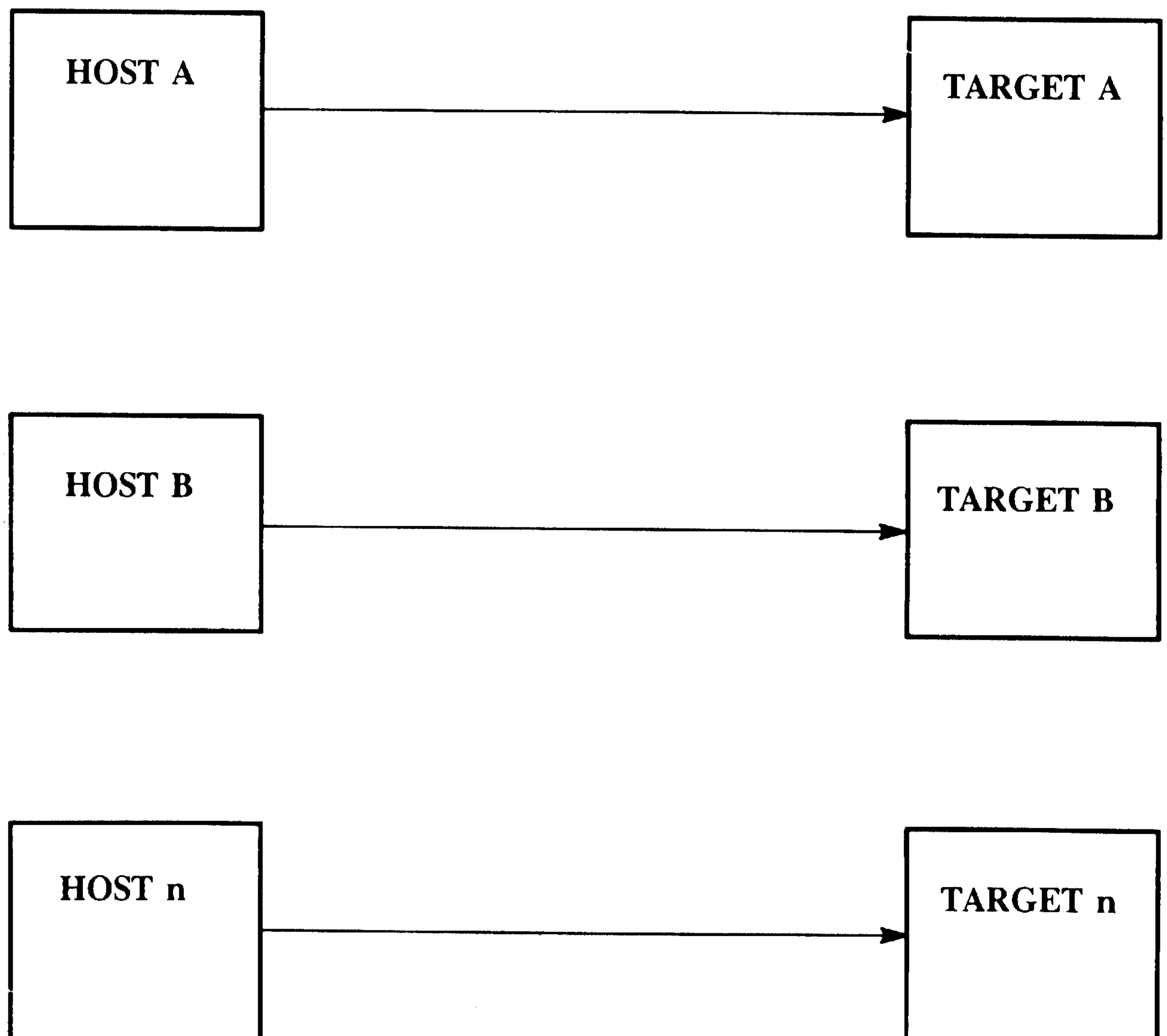
* CROSS-COMPILER EVALUATION:

CONCLUDING REMARKS:

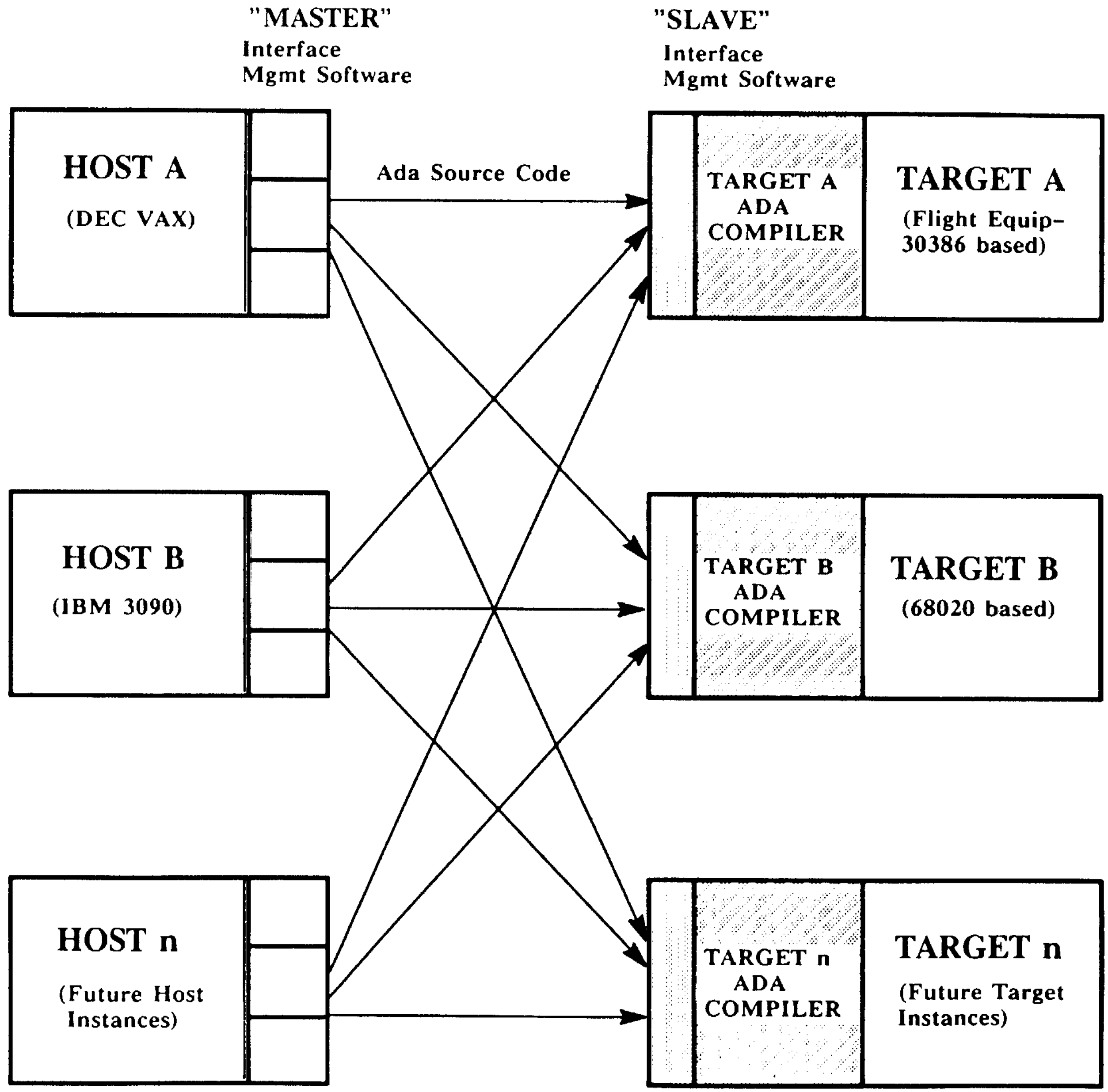
GENERIC PROBLEM STATEMENT



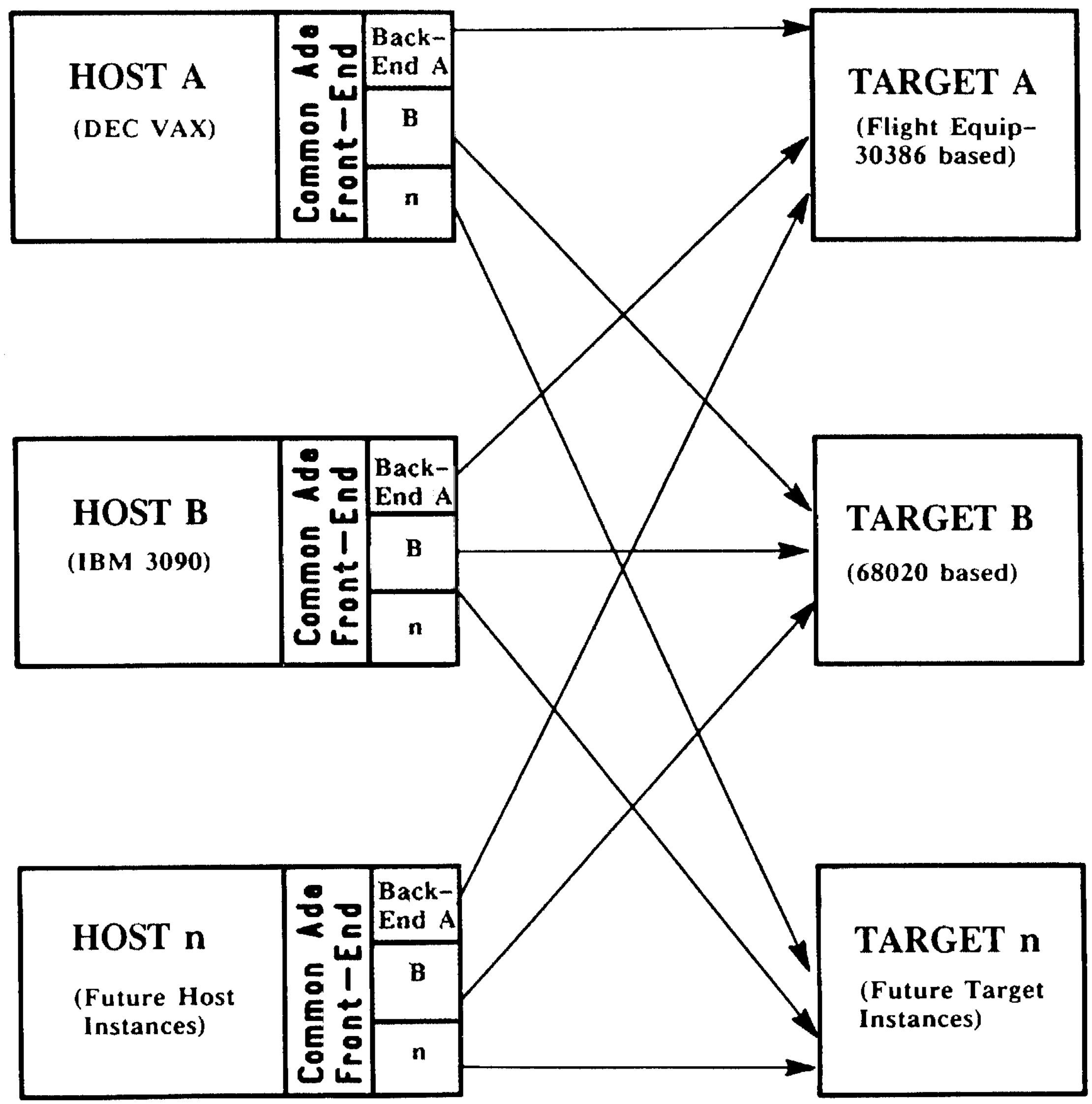
TRADITIONAL "HOST TO TARGET" SOLUTION



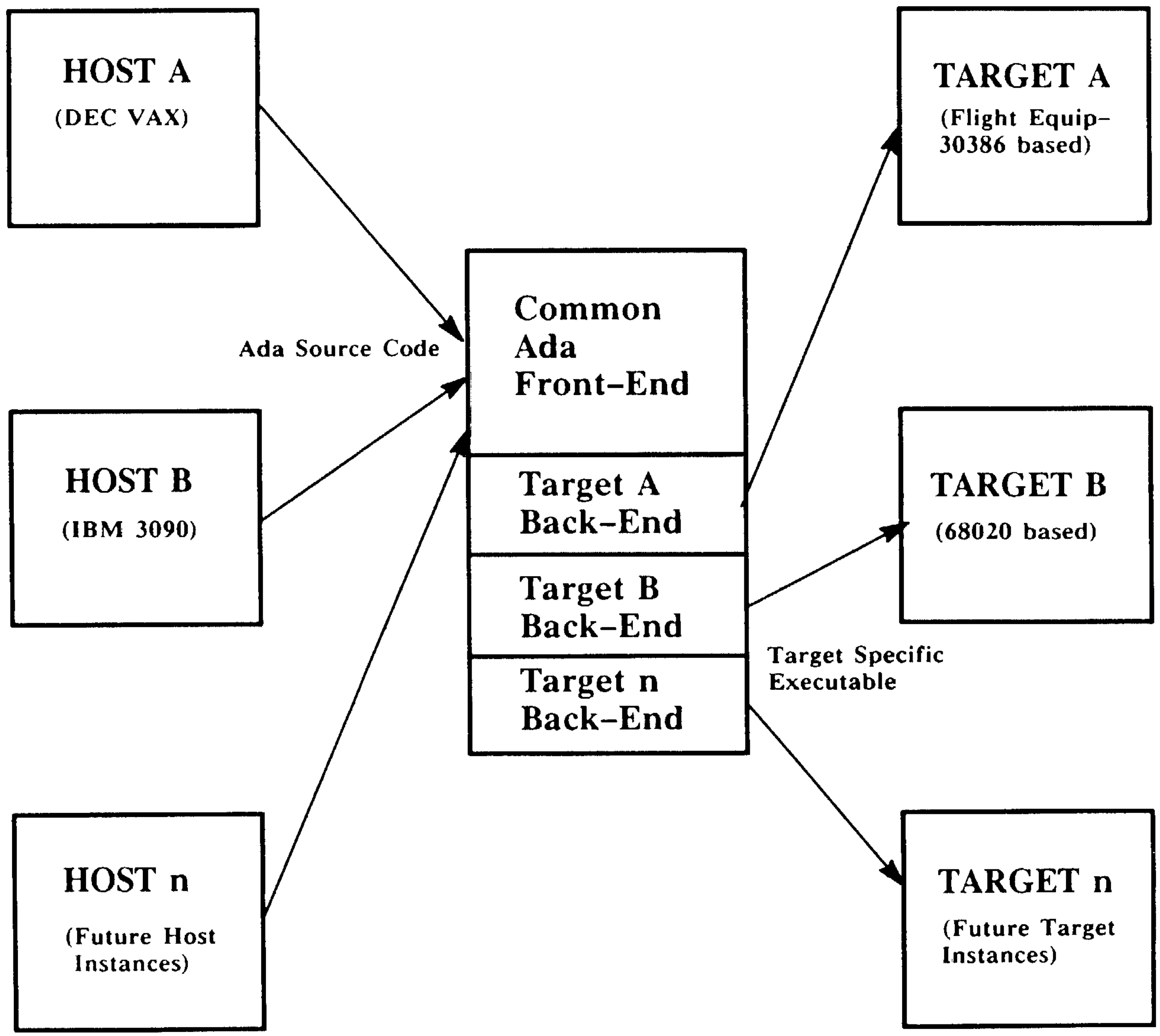
ALTERNATIVE 1



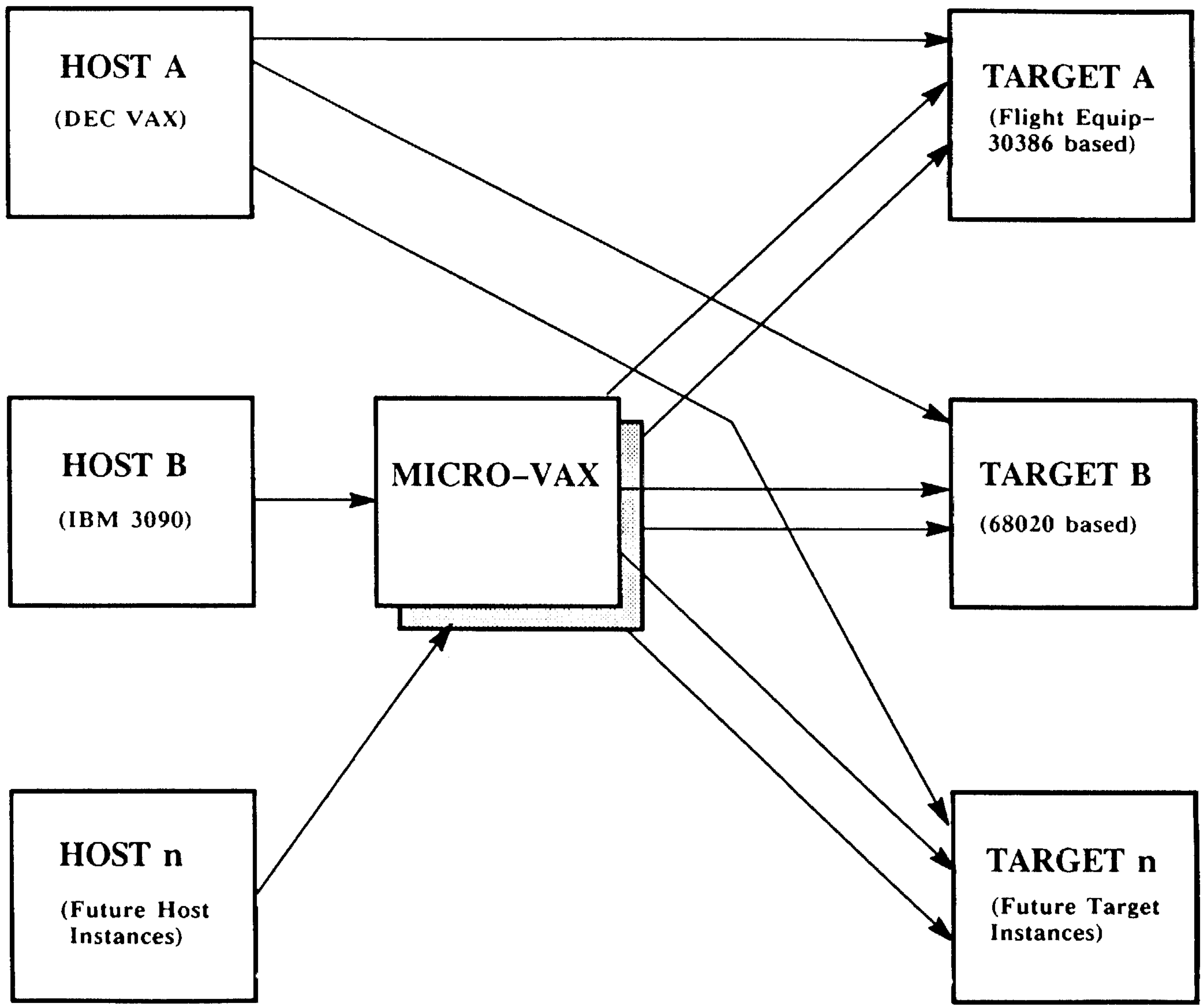
ALTERNATIVE 2



ALTERNATIVE 3



ALTERNATIVE 4



SSE SYSTEM PROJECT

PLAN OF ACTION: SEWG/TEWIG

SEWG (SYSTEM ENVIRONMENT WORK GROUP)

- o JOINT LOCAL EFFORT
- o WHO ARE THEY
- o CROSS COMPILER EVALUATION CRITERIA

SSE SYSTEM PROJECT

PLAN OF ACTION:

HARTSTONE BENCHMARKS

(HARD REAL-TIME)

GENERAL SYSTEM REQUIREMENTS:

- CAPABLE PROCESSOR
- CLOCK SERVICES
- SUPPORT FOR INTERRUPTS

GENERAL BENCHMARK REQUIREMENTS:

- SPANNING HARD REAL-TIME PROBLEM DOMAIN
- INCREASING COMPLEXITY
- STRESS TESTING
- SELF-VERIFYING
- SYNTHETIC WORKLOAD

SSE SYSTEM PROJECT

PLAN OF ACTION: HARTSTONE BENCHMARKS (CONT.)

SPECIFIC REQUIREMENTS:

- o SERIES PH REQUIREMENTS
- o SERIES PN REQUIREMENTS
- o SERIES AH REQUIREMENTS
- o SERIES SH REQUIREMENTS
- o SERIES SA REQUIREMENTS

REFERENCES:

1) A SYNTHETIC BENCHMARK SUITE FOR HARD REAL-TIME APPLICATIONS, AUTHOR: N. H. WELDERMAN, SEI, DATED 17 JUNE, 1988.

2) Ada COMPILER SELECTION HANDBOOK, DRAFT, AUTHOR: N. H. WELDERMAN, SEI, DATED 4 NOVEMBER, 1988.

PLAN OF ACTION: SERC (ARTEWG)

DOCUMENTATION:

- CATALOGUE OF Ada RUNTIME IMPLEMENTATION DEPENDENCIES, 1 DEC., 1987, ARTEWG
- A CATALOG OF INTERFACE FEATURES AND OPTIONS FOR THE Ada RUNTIME ENVIRONMENT, DEC., 1987, ARTEWG
- Ada RUNTIME PACKAGES, DEC., 1987, GSFC, DSTL-88-002
- Ada PROJECTS AT NASA (RUNTIME ENVIRONMENT ISSUES AND RECOMMENDATIONS), JAN., 1988, GSFC, DSTL-88-001
- A FRAMEWORK FOR DESCRIBING Ada RUNTIME ENVIRONMENTS 15 OCT., 1987, ARTEWG
- FIRST ANNUAL SURVEY OF MISSION CRITICAL APPLICATION REQUIREMENTS FOR Ada RUNTIME ENVIRONMENTS 1 DEC., 1987, ARTEWG

CONCLUDING REMARKS

CONTRACTORS, SUBCONTRACTORS
AND VENDORS

LET'S PUSH THE STATE-OF-THE-ART
PLUS 10% TOGETHER.

Session 3: DIRECTIONS AND IMPLICATIONS

1. Robert Nelson, NASA/Space Station Freedom Program Office
2. Glen Freedman, Univ. of Houston at Clear Lake
3. Garry Walker, JPL
4. Frank McGarry, NASA/GSFC

Implications of Ada for Space Station Freedom

Presentation to the NASA Ada Symposium

December 1, 1988

Robert W. Nelson
NASA Headquarters
Space Station Freedom Program Office

PRECEDING PAGE BLANK NOT FILMED

Presentation Outline

Software Management Policies

The Ada Mandate

"Level II" Program Office Role in Software

Focus areas with Ada implications

- **Training**
- **Reuse/commonality**
- **Reviews**
- **Deviations and waivers**
- **Software Support Environment**

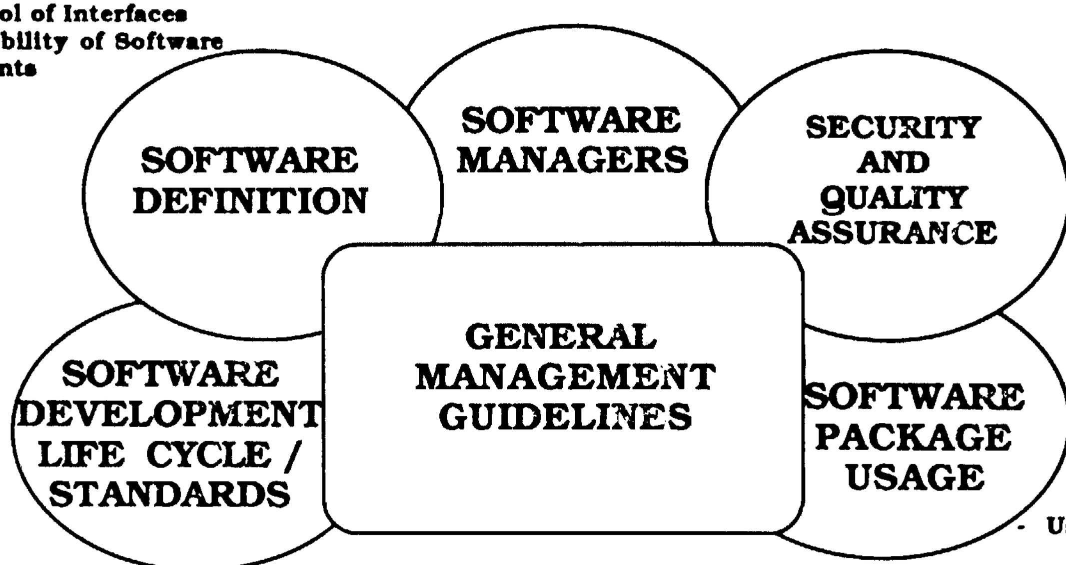
Conclusions

Software Management Policies for the Space Station Program

- Develop Software Categories And Guidance
- Perform Formal Definition And Control of Interfaces
- Ensure Testability of Software Requirements

- Assigned By Each Project Office
- Administer And Implement Software-Related Responsibilities
- Prepare, Maintain And/Or Adhere To A Software Management Plan

- Verify Adherence To Policies, Procedures And Guidelines
- Perform IV&V For Critical Software Elements
- Implement Security Features And Procedures
- Compile Project Data And Lessons Learned



- Follow Standard Software Life Cycle Model
- Adopt Standard Programming Language (Ada)
- Follow Software and Documentation Standards

- Follow NASA Management Instructions And OMB Policies
- Use Life Cycle Management Approach
- Manage Government-Developed and Procured Software In the Same Manner
- Handle Waivers on Case-By-Case Basis

- Use Software Support Environment (SSE)
- Use Commercially Available And/Or Existing Software Consistent With SSE
- Observe Copyright And Commercial Licenses And Obtain Program-Wide Usage/Copying

Rationale for Selection of Ada by the Space Station Program

<u>Only Ada:</u>

- Efficiently supports the most modern software engineering principles
- Is an international and truly standard programming language
- Meets SSP reliability and maintainability requirements
- Can handle the enormous complexity of SSP software
- Will provide SSP with reusable components
- Can be expected to remain economically viable for the life span of SSP

TYPES OF SOFTWARE

- Flight software
  - Distributed systems
  - Element/Payload management
  - International elements
  - OMA
  - Payload user experiment
  - In-flight simulators/trainers
- Ground software
  - Control Center/OMGA (SSCC)
  - Payload planning, scheduling (POIC)
  - Test & checkout (TCMS)
  - User
  - Models
  - Simulations
  - Trainers
  - SSE

PROGRAM OFFICE SOFTWARE ROLES

- Develop and control Space Station Freedom system level software (and hardware) requirements

  ==> System engineering and configuration control part of job

- Ensure that system level software (and hardware) requirements are satisfied by developers

  ==> Monitoring and coordination function

- Ensure that software (and hardware) enable flight elements and distributed systems to perform together and with ground systems as required

  ==> Integration and test part of job

SOFTWARE MANAGER MAJOR RESPONSIBILITIES

Level II Software Manager responsible for Program software process requirements and oversight of technical software requirements.

- Responsible for control of software process requirements
  - Directs common approach for generation of Program-wide software information

-Software Architecture	-Software Schedules
-Software Standards and Methods	-Software Integration
-Software Interfaces	-Software Product Assurance
-Software Sizing and Costing Data	-Software Configuration Management

  - Assures adherence to SSE
  - Develop/control specific software products
    a. Level II Software Policies Document
    b. Level II Software Management Plan
    c. Program Software Standards

- Responsible for establishing end-to-end, Program-wide software requirements perspective.

- Responsible for both processes and technical requirements:
  - Identifies, tracks, and makes sure issues, problems, and risks are addressed. Provides information and recommendations as appropriate to (process/requirements):
    Level II Groups,
    Upper Management, and/or
    Level IIIs
  - Provides point of contact regarding all software issues between Level II, the Internationals, other NASA codes, and SSFP customers.
- Responsible for Software Support Environment Project

Ada and Software Engineering Training

Survey of all NASA Centers in 1987
"A Report on NASA Software Engineering and Ada Training Requirements", November 15, 1987

In General, NASA-wide findings:
- -Less than 25% of project teams responsible for future Ada projects had been exposed to Ada or modern software engineering methodologies
- -Little experience in Ada-related projects for managers; under 6 months for technical

Recommended curriculum for NASA to follow

Space Station Software Managers reported in March 1988 that similar plans and programs for training were available for contractors

Commonality and Reuse

Mandate for commonality across the program

Examples of commonality already achieved:
- Software Support Environment
- User Support Environment (Common User Interface)
- Models and Simulations

Software Support Environment Reuse Library suports the reuse of Ada software utilizing generics and packages

Classification schemes, attributes have been identified

Software Requirements Reviews

Contractors will present the following items for review by SSFPO:

- CSCI hierarchy
- Functional overview
- Performance requirements
- Results of analyses/prototyping
- Requirements traceability
- Qualification requirements
- Quality factor requirements
- Redundancy requirements/planning
- Integration facility support
- Milestone schedules
- Software safety risk assessment
- Software criticality assessment
- Certification planning
- Assembly sequence phasing
- Software Production Facility utilization
- Data flow between each of the functions
- Interface requirements internal/external
- Simulation/model requirements
- User Support Environment utilization
- On-orbit test/verification plans
- Intersite deliverables
- Updates to previous deliverables
- Actions/procedures deviating from plans
- Maintenance approcach
- Requirements for alternate software
- Commercial-off-the-shelf (COTS) usage
- Automation/robotics planning
- Growth planning
- Commonality/reuse

The Software Requirements Reviews will provide insight into software management and compliance with program policies.

Deviations and Waivers from Use of Ada

Draft Policy Being Reviewed by Software Managers

General Information Required:

- Problem statement
- Proposed alternative description
- Comparison of approaches (approved vs alternative)
- Ripple effect of deviation/waiver adoption on other elements

Specific Information Required:

- Rationale and detailed support analysis indicating approved method deficiencies
- Benefit of adoption of alternative method(s)
- Performance, schedules and other data relating to use of alternatives
- Technical impact/benefit of alternative
- Life-cycle cost of implementation of functions with alternative
- Operations/utilization impact of use of alternative

No requests for deviations or waivers from Ada have been made to the Program Office

Conclusion

The SSFPO has an unwaivering committment to the maximum utilization of Ada throughout flight, ground and support software applications.

NASA Ada User's Symposium

NASA/GSFC
December 1, 1988

Software Engineering and Ada Training at NASA/JSC: Myths, Lessons Learned, and Directions

Dr. Glenn B. Freedman, Director
Software Engineering Professional Education Center
University of Houston - Clear Lake
Houston, TX 77058
(713) 488-9433

Software Engineering and Ada Training at NASA/JSC

- The Myths
- The Lessons Learned
- The Directions

An Introductory Editorial

"

The only thing more expensive than education is ignorance. -- Benjamin Franklin

Change is inevitable. In a progressive country, change is constant. -- Benjamin Disraeli

Change is not made without inconvenience - even from worse to better. -- Richard Hooker

Human history becomes more and more a race between education and catastrophe. -- H.G. Wells

"

Background

Definition of the Population

Job Responsibilities

Levels of Knowledge/Skill/Attitude

Definition of Software Engineering

Knowledge

Activities

Definition of the Environment

Computing Environment (H/I/T)

Scale of the Project (S/M/L/LCDNF)

Education/Training (E/T)

Myth #1

The Myth: Management support alone will ensure an orderly transition to software engineering with Ada technology.

The Lesson Learned: Management support is a necessary, but not the sole prerequisite to success. Difficulty in communication about software both vertically and horizontally in the organization. We should have a consistent management message and build that message into the curriculum.

Myth #2

The Myth: Unlimited funding will ensure success.

The Lesson Learned: It is not how much money is spent, but how money is spent, after a certain minimal level of support is reached. We should use tools of project management, educational leveraging, and an integrated educational program to optimize the limited resources.

Myth #3

The Myth: Techies know best.

The Lesson Learned: When it comes to software, perhaps so. When it comes to education, not so. We should build a sound education/training program that evolves from a cooperative union among government, universities, and industry.

Myth #4

The Myth: Ad Hocracy works in education/training.

The Lesson Learned: E/T is as much a part of the software life cycle as configuration management or requirements analysis. Lack of either often results in unmaintainable software systems. We should build E/T into the life cycle systematically. It can be done.

Myth #5

The Myth: There is sufficient, quality software engineering training available.

The Lesson Learned: There isn't. There is a supply of Ada trainers, but woefully few experienced, knowledgeable software engineers who are also decent teachers. This means that we should use our own resources, groom our own, and refine our own.

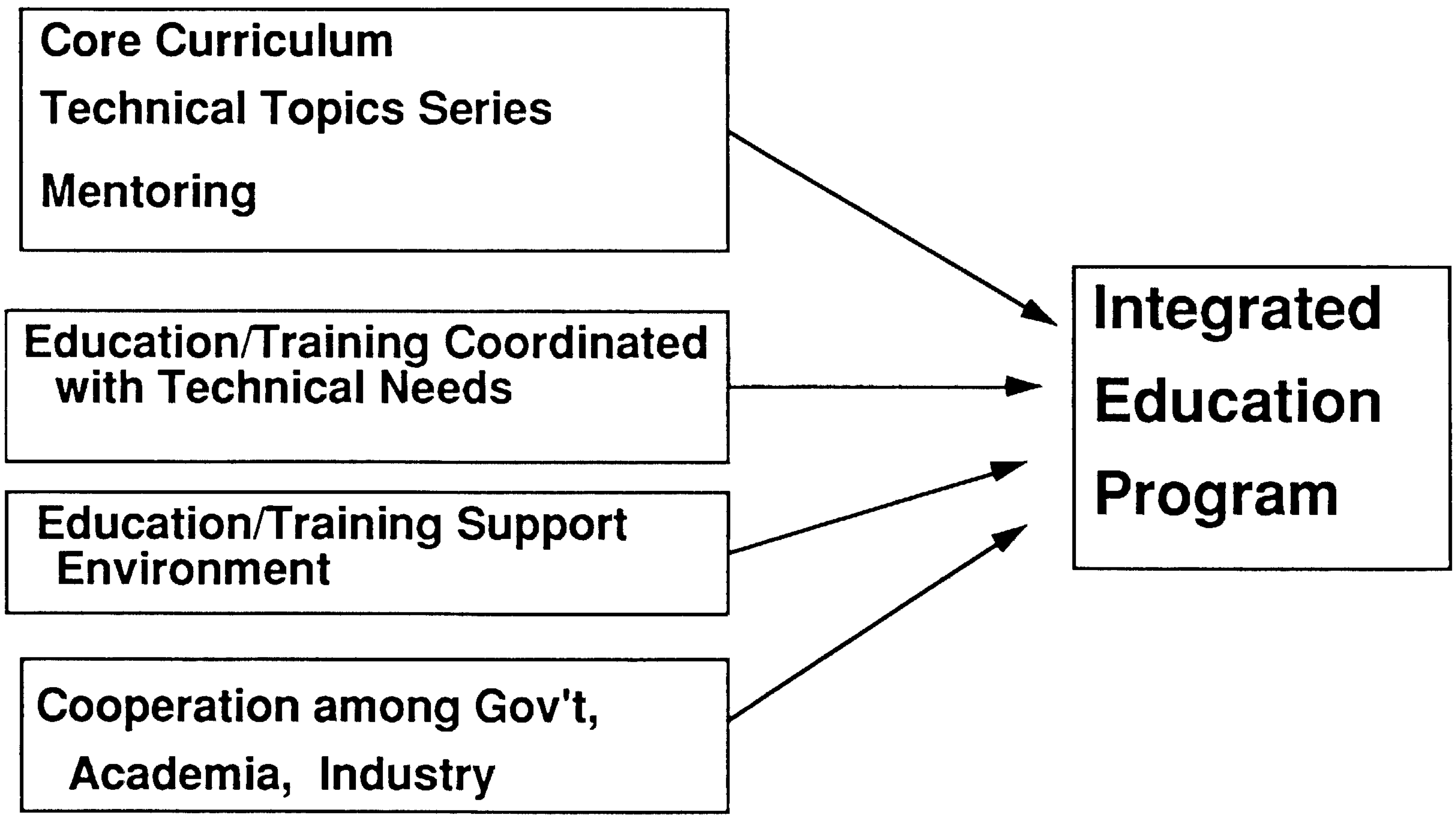
Myth #6

The Myth: Everyone is committed to E/T in software engineering.

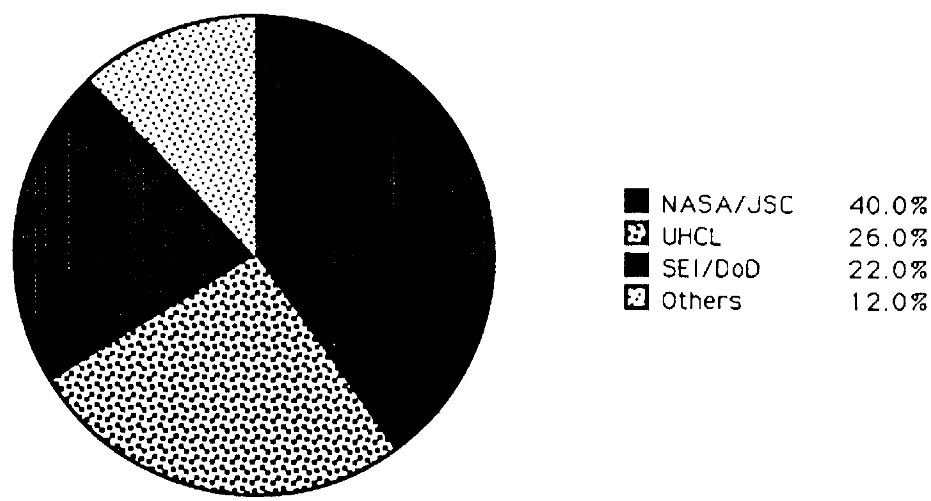
The Lesson Learned: Motherhood and apple pie. Unfortunately, too often commitment, like beauty is only skin deep. Before crunch time*, we heard that it was too early to educate or train. When crunch time began, we often heard that it was too late. We should build E/T into the life cycle through the notion of preparatory, iterative sets.

*Crunch Time = The inevitable point in software time and space when the software producer realizes that the cost and/or schedule won't be met - and software consumer knows it.

Directions


Core Curriculum
Technical Topics Series
Mentoring
Education/Training Coordinated with Technical Needs
Education/Training Support Environment
Cooperation among Gov't, Academia, Industry
Integrated Education Program

Credits Where Credits are Due



The opinions expressed herein are solely those of the author and are not the official position of any agency.

JPL
Ada Development Laboratory

The Jet Propulsion Laboratory: Transition to Ada Software Development

Gary N. Walker
1 December 1987

Catalysts for creation of JPL Ada Development Laboratory:

Limited JPL experience with Ada

Global Decision Support System
-- Command and Control System for Military Airlift Command
-- 279K Lines of Ada Code (374 L.O.C. with comments, etc.)
-- 12 - 15 Subcontractors
-- Interfaced with RDB and GKS through Fortran, C, and Macro

JPL commitment to software development improvement

SSORCE burden funded software development organization
-- SORCE to sharpen software engineering methodologies and standards
-- SERC to support systems engineering and system management
-- SPARC to support software product assurance programs
-- SI&TRCE to support systems integration and test
-- OPERC to support operations engineering

JPL's need to keep in step with technology and sponsors' needs.

-- Ada support for current software engineering methodologies
-- Increasing number of NASA, FAA, and DoD Ada directives

JPL management realized that better tools are required.

-- Save money
-- Save time
-- Improve consistency
-- Improve quality

A centralized JPL Ada Development Laboratory intended to:

-- Provide Ada tools for development

Lack of tool continuity: Most JPL work is done on a project basis. Projects procure equipment and software tools necessary for a given work unit. In most cases, tasks return tools as deliverables.

Lab management decision to make institutional commitment to a centralized facility to benefit a wide spectrum of tasks and provide for continuity.

-- Train and educate JPL personnel

-- Provide a testbed for metrics study

-- Provide a source of consultation assistance

-- Promote Ada and software engineering practices (users' group, etc.)

Training and Education:

	Educate	Train
Management	X	
Sponsors	X	
Architects & Engineers	X	X
Programmers	X	X

Training includes developing proficiencies in the use of Ada, software engineering tools, and environments.

Education includes:
-- What are *"good"* software engineering practices?
-- What Ada is?
-- What Ada is not?
-- What Ada will do for development?
-- What Ada will do to development?

Staff Development:

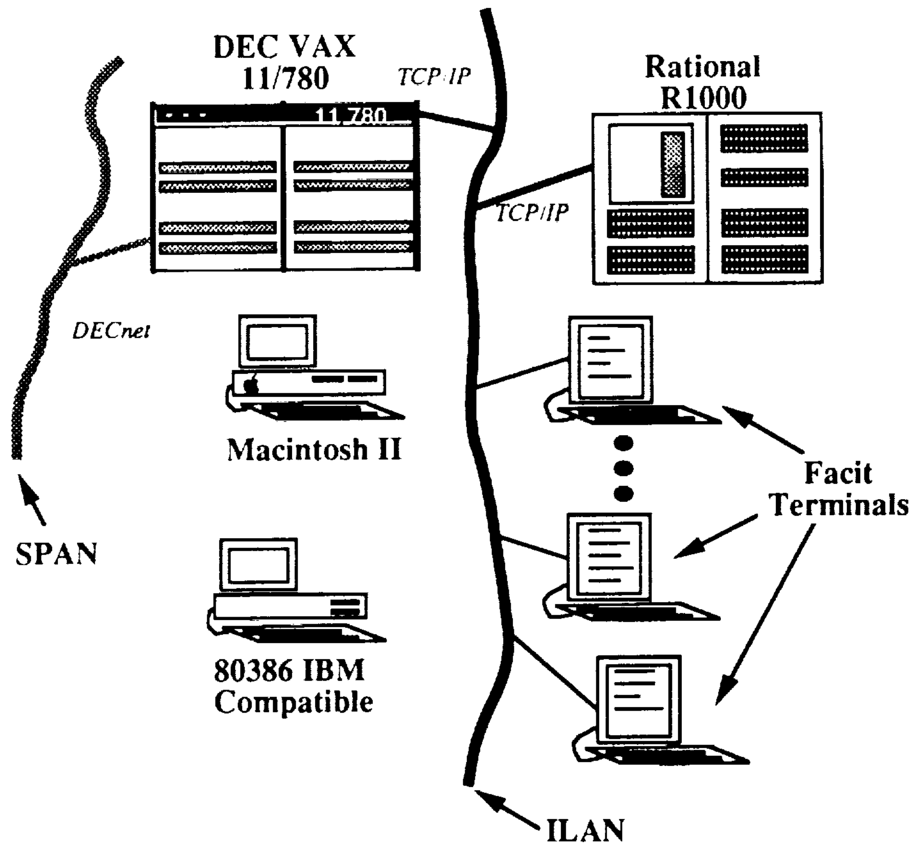
Training

-- Rational Fundamentals
-- Advanced Topics
-- Basic Subsystems and Configuration Management
-- Networking
-- Design Facility
-- Target Build Facility
-- Cross Development Facility
-- Project Design Methodology
-- Ada fundamentals

Education

-- Management class
-- Seminars on concept of dealing with Ada

ADL Facility and Equipment Suite:



1300 Sq. Ft. development center being built

Reuse of an existing VAX

Institutional purchase of Rational R1000 Model 20

Microcomputer equipment support
- -- Design tools
- -- Ada compilers
- -- Ada tutorials

Rational:

The Rational Ada Development System

- Validated Ada® compiler
- Ada-specific productivity tools
- Networking compatibility with ILAN and TCP/IP
- Configuration management and version control
- Workorder/change tracking
- Statistics collection
- Standardized documentation generation
- A user/vendor customizable user interface

Rational:

Advantages of Using a Universal Host Environment

-- High degree of parallelism can be built into schedules

-- Selection of the host hardware architecture and operating system can be delayed

-- Training and tool development in a common environment

-- Project managers are more flexible to move staff among tasks for different targets

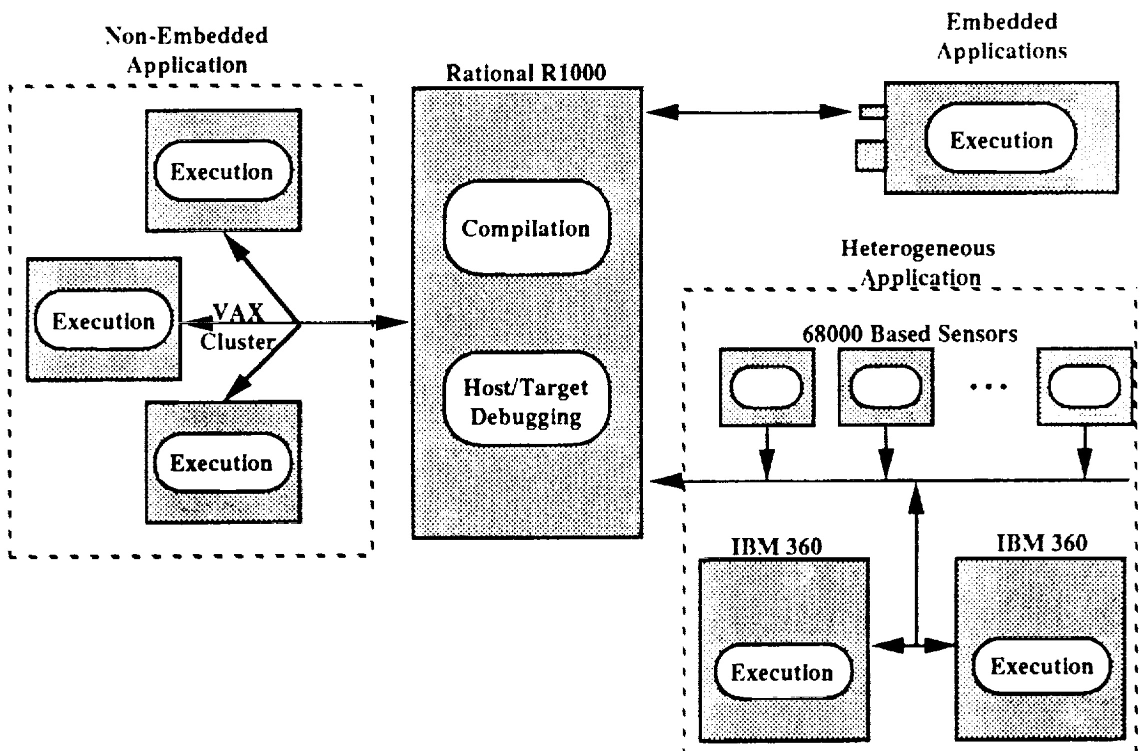
-- Tools have permanence and are reuseable

-- Incremental compilation provides rapid turnaround

-- Host/target debugging uses universal host environment while working on target

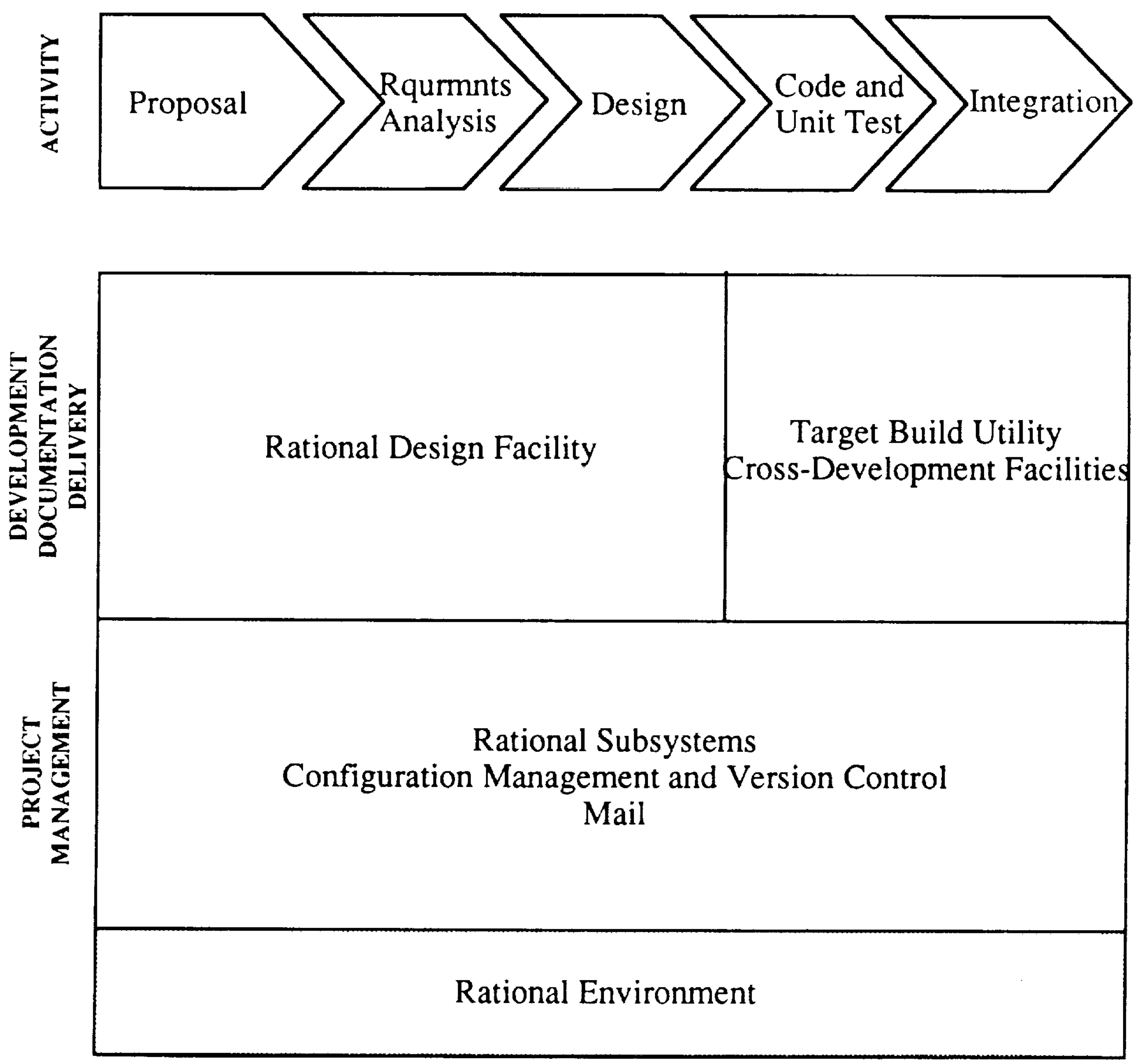
-- Common and host specific code are manageable in the same environment

Universal Host Development



Rational:

The Software Lifecycle and Rational Tools



Current Ada Activities at JPL:

Network Operations Communications Center Upgrade

-- Development on Rational and VAX
-- Target Host is to be determined

Ground Communications Facility Upgrade

-- Development on Rational and Gould
-- Target Host is Gould

ASAS/ENSCE

-- Development on Rational and VAX
-- Target Host is VAX

Realtime Weather Processor

-- Development on VAX
-- Target Host is VAX

Problems to be Addressed:

Manpower

-- Hiring
-- Maintaining
-- Training

Who should purchase?

VAX Type	Ada Compiler	VAXSET Ada Environment
μVAX II	$ 15.7K	$ 16.4K
11/780	$ 31.7K	$ 33.1K
8600	$ 57.5K	$ 60.2K
8800	$ 70.6K+	$ 98.6K+

For what work is Ada appropriate?

What Ada features should be used?

How should compiler compatibility be studied?

How should tool development be funded?

How should reuse libraries be maintained?

Experiences With Ada at NASA/GSFC

Implications and Directions

FRANK MCGARRY
NASA/GSFC

Ada/PROMISES

• Increased Productivity	(at least lower life cycle cost)
• Higher Reliability	
• Software Engineering Practices	Abstraction Strong typing Information Hiding
• Commonality	Language Across Environments Tools Methods
• Portability	Software and People
• Improved Maintainability	
• Increased Management Visibility	

EXPECTATIONS OF Ada

What Problems Does it Address

Ada Does Not Address

Technological Difficulties*

Ada Does Address

ESSENTIAL DIFFICULTIES

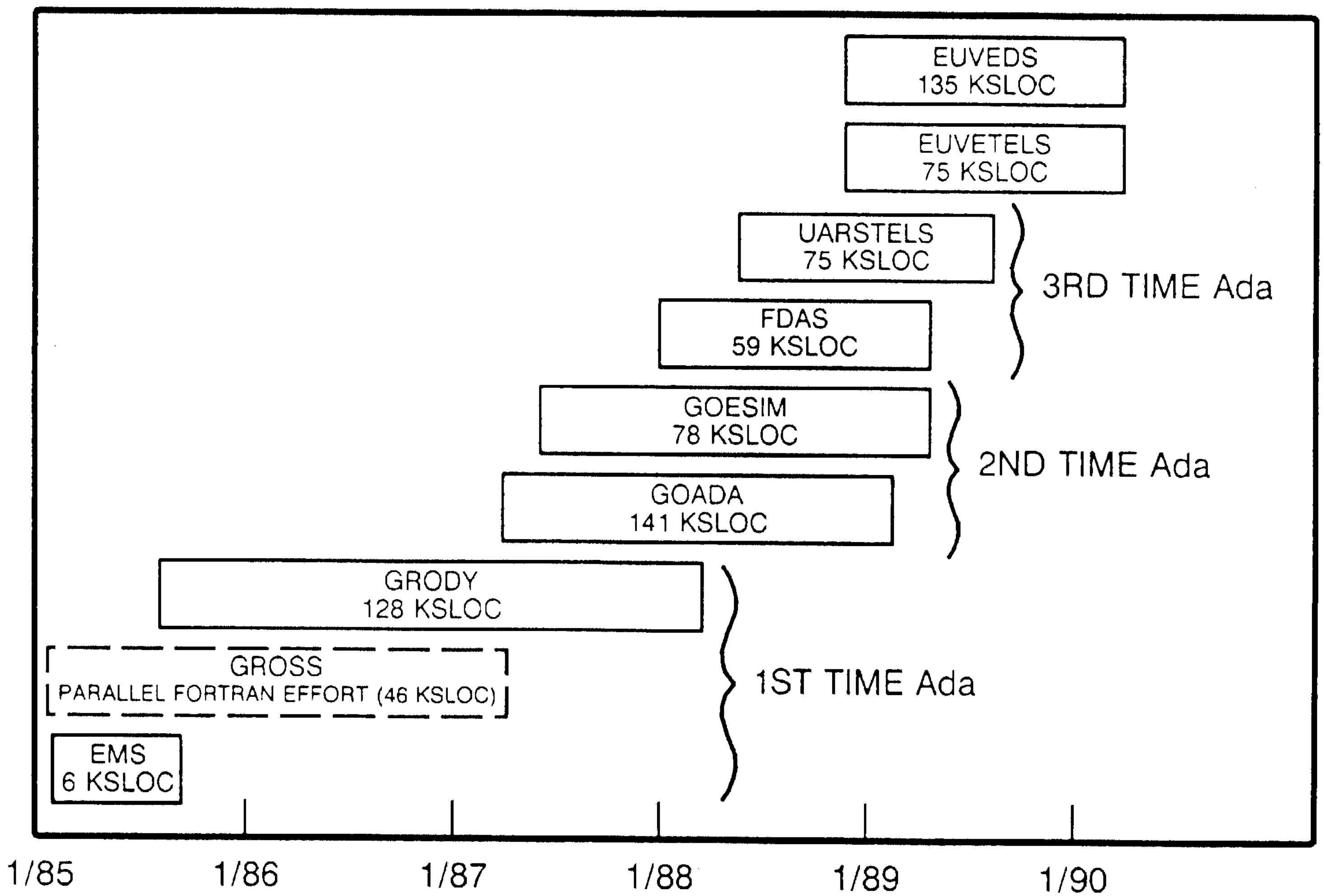
- Complexity
- Conformity
- Changeability
- Invisibility

ACCIDENTAL DIFFICULTIES

- Language Complexity
- Resource Limitations
- Diversity of Environment and Languages

* Fred Brooks "No Silver Bullet"

Ada PROJECTS IN FLIGHT DYNAMICS DIVISION



D217.003

SOFTWARE CHARACTERISTICS

	GROSS	GRODY	GOADA	GOESIM	FDAS	VARSTELS	TYPICAL TM SIMULATION
	(FORTRAN)	(1ST TIME Ada)		(2ND TIME Ada)	(3RD TIME Ada)		FORTRAN
TOTAL LINES (CR)	45500	128000	139000	78000	58700	75000	28000
NON COMMENT/ NON BLANK	26000	60000	68500	36000	31300	-	15000
EXECUTABLE LINES (NO TYPE DECL)	22500	40250	42000	21000	17100	-	12500
STATEMENTS (SEMI-COLON INCLUDES TYPE DECL)	22300	22500	25000	14000	11000	-	12000
% REUSED	36%	0	38%	32%	NA	42%	15%

1. Ada RESULTS IN LARGER SYSTEM (SLOC)
2. REUSE TREND VERY POSITIVE
3. "LINE OF CODE" DEFINITION CRITICAL

D217 006

Ada IMPACTS ON LIFE CYCLE EFFORT DISTRIBUTION

% TOTAL EFFORT*

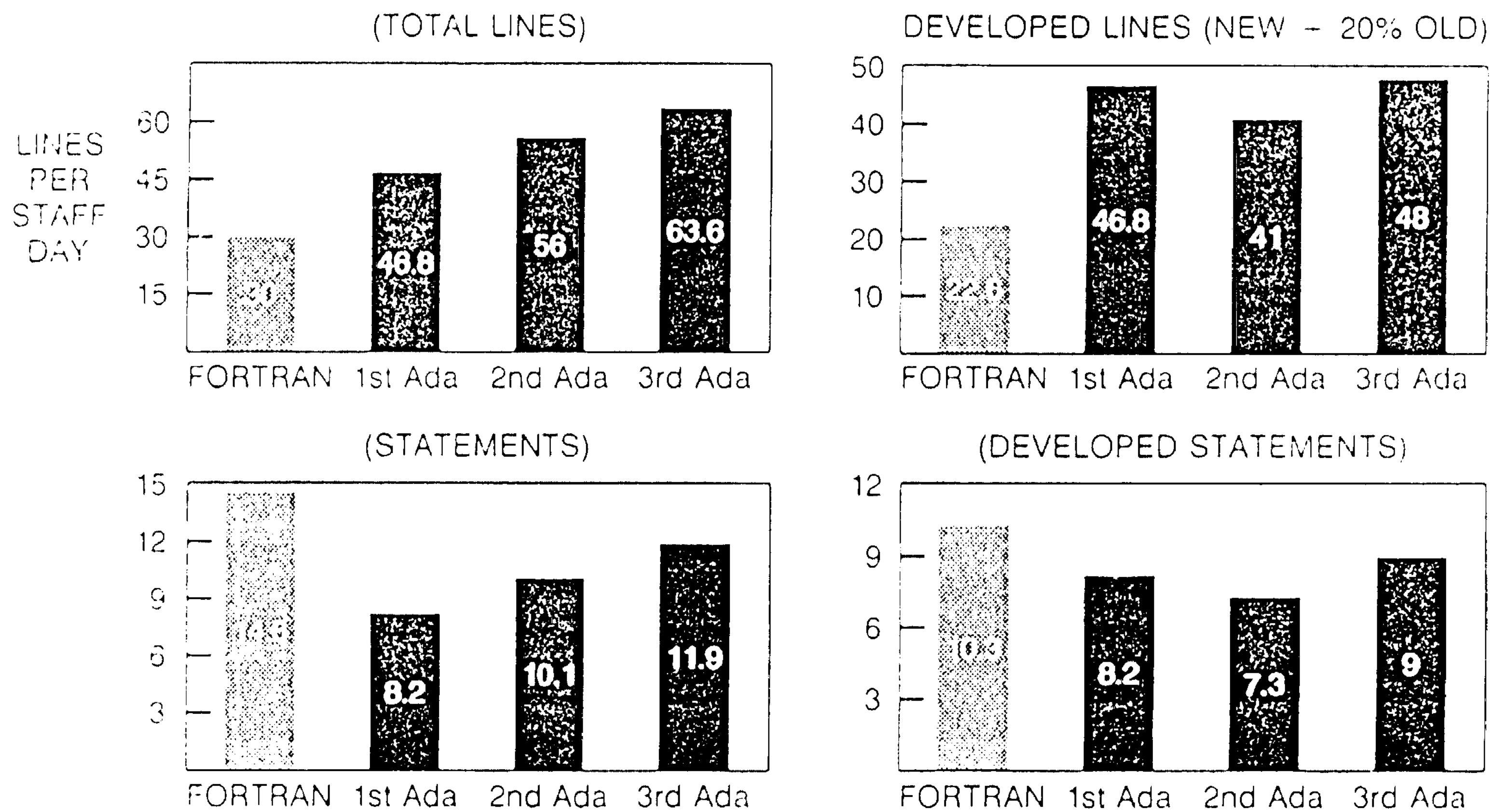
	GROSS (FORTRAN)	GRODY (1ST TIME Ada)	GOADA (2ND TIME Ada)	GOESIM (2ND TIME Ada)	FDAS (3RD TIME Ada)	VARSTELS (3RD TIME Ada)
PRE DESIGN	8	8	6	4	8	-
DESIGN	27	24	32	34	34	-
CODE	40	42	42	41	38	-
TEST	25	26	20	21	20	-
TOTAL EFFORT (HOURS)	12150	21860	21230**	10360**	7390**	-

SIGNIFICANT CHANGES TO LIFE CYCLE
HAVE NOT YET BEEN OBSERVED -
BUT . . .

*EFFORT DISTRIBUTION BASED ON PHASE DATES (E.G. END DESIGN, END CODE, END TEST)
** PARTIALLY BASED ON ESTIMATES TO COMPLETION

D217 007

Ada COST/PRODUCTIVITY



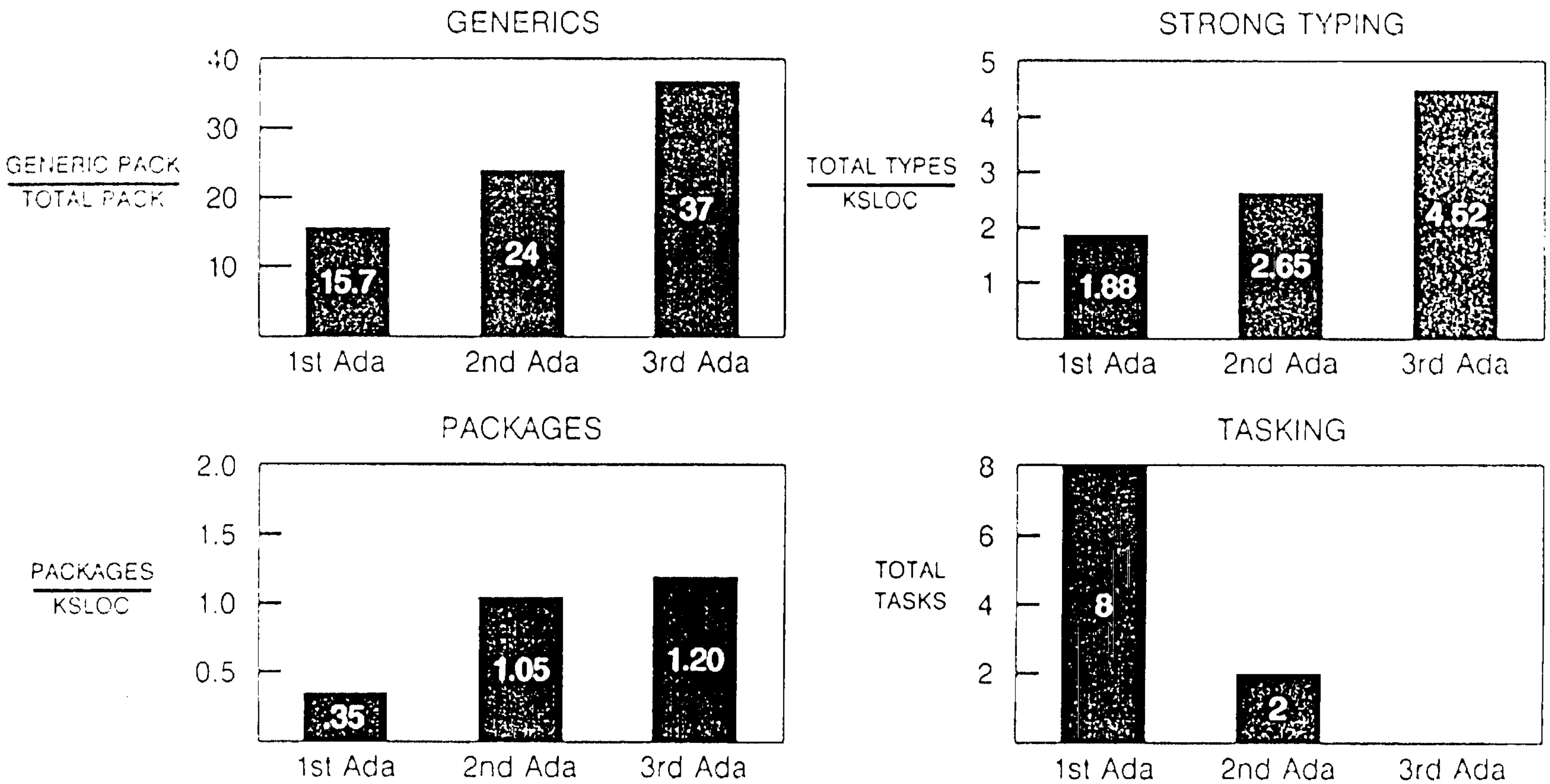
- CLEARLY DEFINE "LINES OF CODE"
- DO NOT USE SLOC IN COMPARING FORTRAN/Ada
- Ada TRENDS ARE IN POSITIVE DIRECTION

(GROSS/GRODY/GOADA/GOESIM/FDAS)

ORIGINAL PAGE IS OF POOR QUALITY

D217.016

USE OF Ada FEATURES



- USE OF Ada FEATURES CHANGES APPRECIATELY WITH EXPERIENCE
- NOT ALL FEATURE APPROPRIATE FOR APPLICATION

D217.018

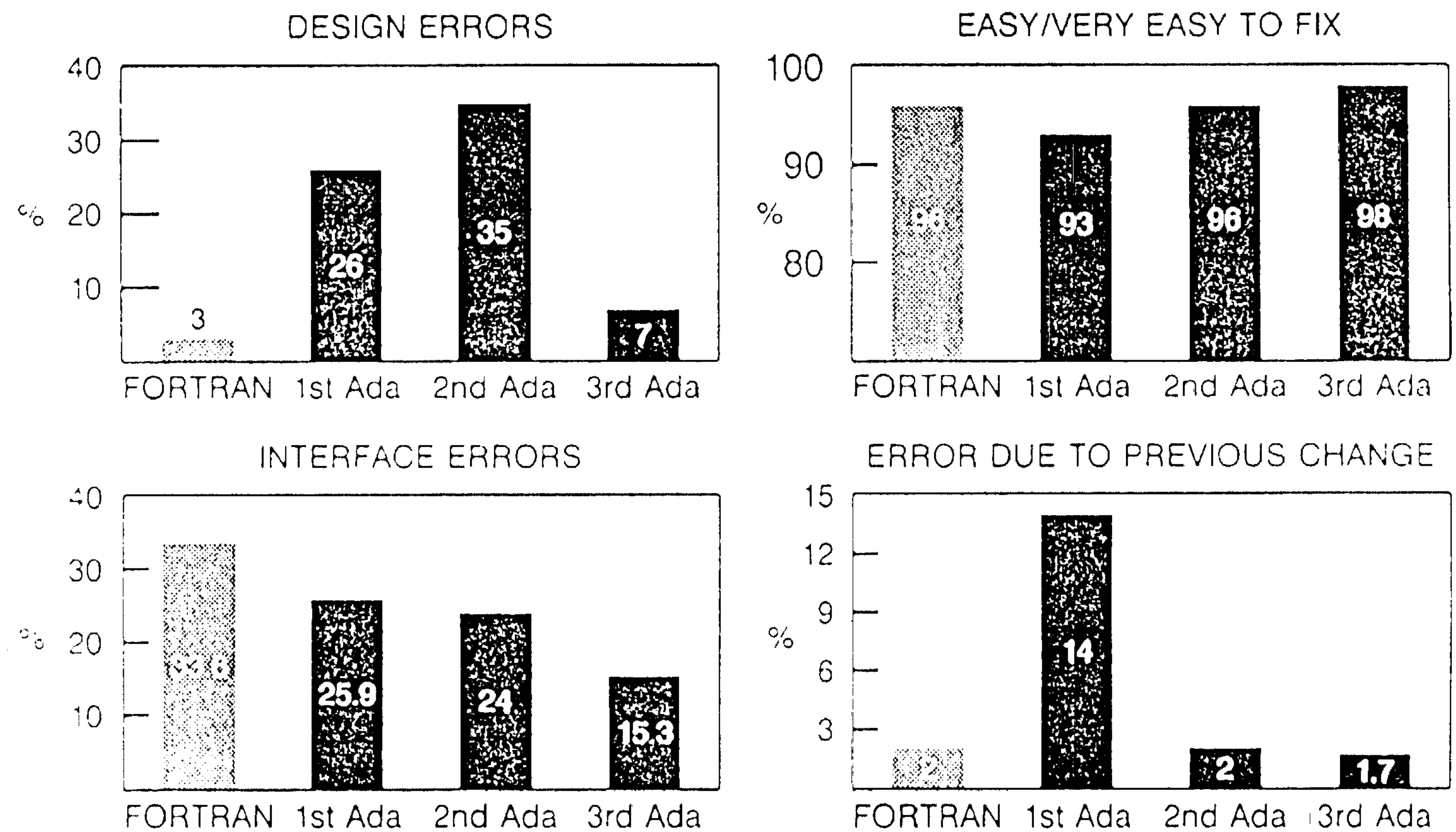
Ada AND ERROR/CHANGE RATE

	GROSS (FORTRAN)	GRODY	GOADA	GOESIM	FDAS
CHANGES/KSLOC*	5.8	4.2	2.8	2.4	6.8
ERRORS/KSLOC	3.4	1.8	1.7	1.4	1.0

- RELIABILITY OF Ada SOFTWARE - AT LEAST AS GOOD AS FORTRAN
- VERY POSITIVE TRENDS FOR Ada - OVER TIME

*SLOC = TOTAL LINES (INCLUDES COMMENTS/REUSED)

ERROR CHARACTERISTICS

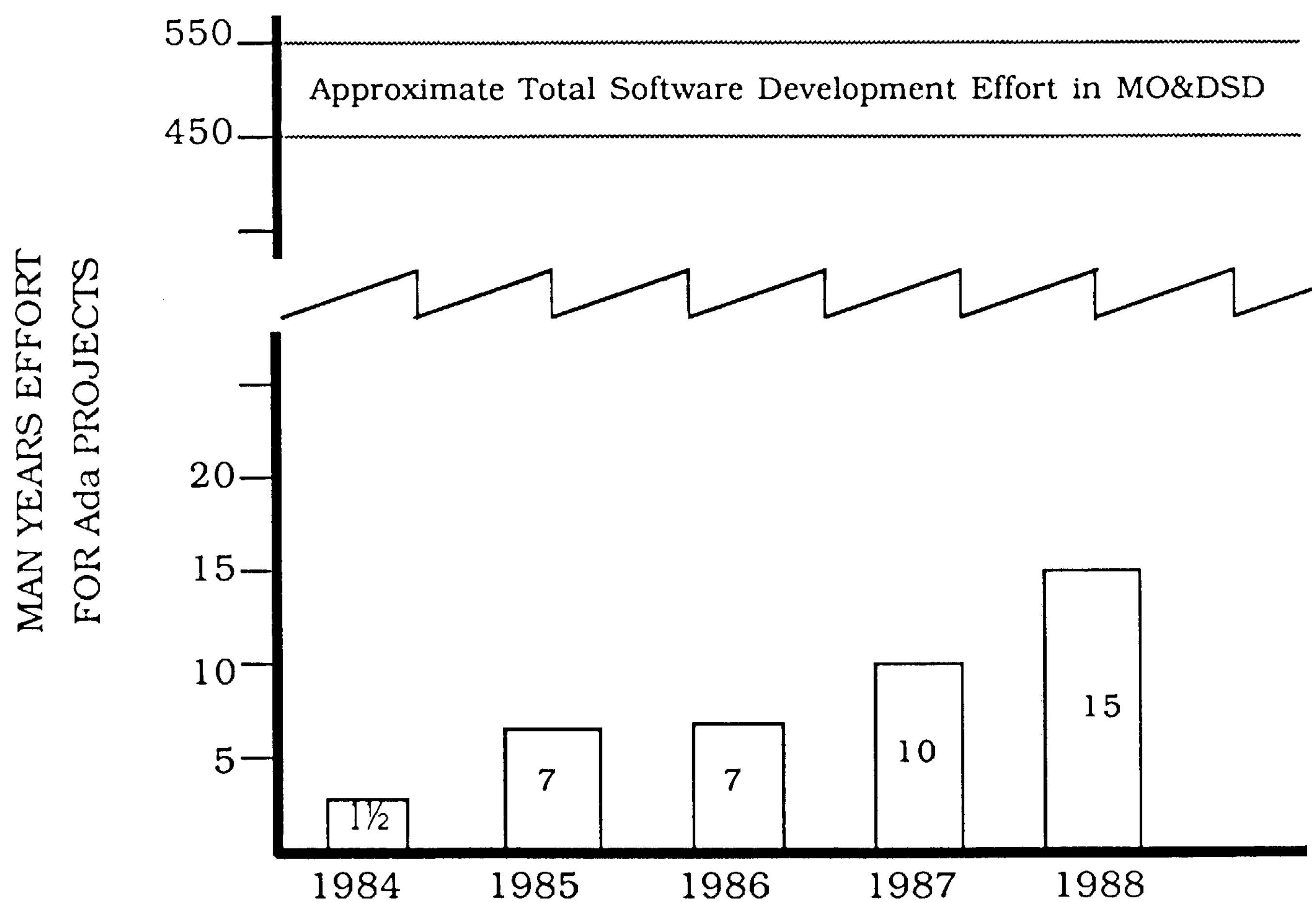


- Ada ERROR PROFILE CHANGES WITH MATURITY OF USE
- Ada HELPS CUT INTERFACE ERRORS

D217.017

C-3

Ada SOFTWARE DEVELOPMENT EFFORT IN MO&DSD



Experience Base is Small and Growing Too Slowly

EVOLVING IMPACTS OF Ada

OBSERVATIONS
(FROM 6 PROJECTS IN THE SEL)

1. COST	• 30% + OVERHEAD TO 'FIRST TIME' PROJECTS • SIGNIFICANT INPROVEMENTS ON 2ND TIME/3RD TIME USE
2. RELIABILITY	• INITIALLY SIMILAR TO FORTRAN • IMPROVEMENTS WITH EXPERIENCE
3. REUSE	• VERY POSITIVE TRENDS - EXCEEDS FORTRAN EXPERIENCE EARLY
4. SIZE	(ADA TO FORTRAN - LARGER) • TOTAL LINES 3 TO 1 • EXECUTABLE LINES 2 TO 1 • NON COMMENTS 2 1/2 TO 1 • STATEMENTS 1 TO 1
5. USE OF ADA FEATURES	• PROMINANT EVOLUTION WITH 'EXPERIENCE' • SEEMS RELATED TO IMPROVED DEVELOPMENT • SOME FEATURES INAPPROPRIATE TO SPECIFIC PROBLEMS
6. EXPERIENCE BASE	• EVOLUTION TO ADA (FROM STANDARD FORTRAN) - 10 YEARS + • CURRENT EXPERIENCE BASE IS GROWING TOO SLOWLY

IMPLICATION/OBSERVATION
(GENERALIZING TO NASA)

1

Evolving to Ada
(Staffing Needs)

NASA DOES NOT HAVE ADEQUATELY TRAINED/EXPERIENCED PERSONNEL

• <u>TRAINING</u>

1. SPACE STATION STUDY (10/87 BY SOFTECH)	- EXCELLENT FOUNDATION FOR REQUIRED Ada TRAINING
2. CURRENT Ada TRAINING PROGRAM	- 'SHOT GUN APPROACH' • UNCOORDINATED • NO FOLLOW-UP • MINIMAL ASSESSMENTS
3. PLANNED TRAINING PROGRAMS	- NO IMPROVEMENTS SEEN

• <u>EXPERIENCE</u>

1. CURRENT Ada EXPERIENCE BASE	- INADEQUATE (IF NASA IS SERIOUS ABOUT Ada)
2. PiLOT PROJECTS	- INADEQUATE TO SUPPORT TRAINING FOLLOW-UP
3. PLANS FOR BUIDLING BASE	- MINIMAL (LOOKING TOWARD OJT)

IMPLICATIONS/OBSERVATIONS
(Generalizing to NASA)

2

- <u>Measuring Ada</u>
  - Very Little Evidence of Ada Impact
  - Too Few Attempts at Measuring/Baselining
  - Do We Know Where We Are?
    (Cost/Reliability/Strengths/Weaknesses/ . . .)

- <u>Infrastructure</u>
  - Excellent Potential in SMAP/NISE, . . .
  - 'Today' is First Attempt to Coordinate Status/
    Direction . . . Across NASA
  - Policies/Guidelines Evolving
  - Environments Evolving
  - No "Process Assessment" Mechanism

> • Critical need for Software Measurement
> • NASA Infrastructure Heading in Supportive Direction

IMPLICATIONS/OBSERVATIONS

3

EVOLVING TO Ada PLANNING

THE NEED	PRELIMINARY OBSERVATIONS
• TRAINING/DEVELOPMENT PLANS	• EXCELLENT TRAINING PLAN VIA SPACE STATION (SOFTECH REPORT)
	• INADEQUATE DEVELOPMENT PLANS
• RISK MANAGEMENT	• NONE IDENTIFIED
• CONTINGENCY	• NONE IDENTIFIED
• MEASURES FOR ASSESSMENT	• NONE IDENTIFIED

PLANNING FOR 'Ada' HAS BEEN INADEQUATE

EVOLVING TO Ada
EXPERIENCES IN NASA

- SOFTWARE ENGINEERS BELIEVE Ada IS THE VEHICLE FOR IMPROVED SOFTWARE

BUT

1. NO EVIDENCE DEMONSTRATING VALUE OF Ada (S/W COST/RELIABILITY/ . . .)
2. STILL MUCH DEBATE ON TRAINING/IMPACTS/APPLICATIONS/BENEFITS/ . . .
   (TOO MANY STUDIES - NOT ENOUGH PRACTICAL DEMONSTRATION)
3. EXCESSIVE HAND WAVING AND PREMATURE PROMISES

Ada IS YET AN UNKNOWN, UNPROVEN TECHNOLOGY IN NASA (AND EVERYWHERE ELSE)

DIRECTIONS WITH Ada

Ada IN CODE 500
POINTS TO CONSIDER

1. Ada is here & it will stay
2. Software Development (for Code 500) will be better off by evolving toward Ada (Training/environment/awareness/commonality . . .)
3. Currently - No NASA Policy for Ada
4. Ada (today) Cannot Support Time-Critical system nor Production Systems on many environments (in MO&DSD)
5. Environment/training will Support Ada Concepts (e.g. SSE/CSSE . . .)
6. History Shows 16 to 20 years required for Software Technology 'Insertion'*
7. Ada Adds Significant Development Cost - During 'Adaptation' Phase

* Riddle/Redwine Report for STARS ('84)

DIRECTIONS WITH Ada IN NASA

- Adopt Specific 'POLICY' in Ada . . . (Ada's Role in NASA)
- Expand 'PILOT' Development Efforts Across Agency
- Formulate Broad Software 'MEASUREMENT' Program
- Increase 'EXPERIMENTATION'/Study/Refinement (What does Ada Imply?)
- Generate 'INCENTIVE' Program with Support Contractors (Use/training . . .)
- Expand Role of 'SSE" (concepts) beyond Space Station
- Restructure 'TRAINING' (Ada/SE) Approach in NASA
- Modify Support 'INFRASTRUCTURE'
  - Process Assessment Team(s)
  - Software Engineeirng /Ada Adaptation Team(s)
  - SMAP - (Increase Role)

Appendix A:

OPEN DISCUSSION

PRECEDING PAGE BLANK NOT FILMED

OPEN DISCUSSION

Moderator
Ed Seidewitz, Goddard Space Flight Center

Panelists
Gary Walker, Jet Propulsion Laboratory
Michael Holloway, NASA Langley Research Center
William Howle, NASA Marshall Space Flight Center
Frank McGarry, NASA Goddard Space Flight Center
Robert Nelson, NASA Space Station Program Office
Kathy Rogers, MITRE (for NASA Johnson Space Center)

Recorder
Dwight Shank, Computer Sciences Corporation

The final session of the symposium provided the opportunity for an active, open discussion between the audience and panelists representing various NASA centers. The following is not a transcript of the session, but is instead an attempt to summarize some key points addressed during the discussion. These points are organized into broad areas which reflect the general themes which emerged during the course of the symposium.

Transition

There are both management and technical issues involved in the transition to Ada. The panel was asked to address the issue of managing the risk of transition. Bob Nelson remarked on the need for a risk management approach and on the management of risk at the project as well as the organizational level.

There were also comments from the audience on specific projects which addressed risk management. Eileen Quann of Fastrak Training mentioned that risk management was an important consideration in the decision to use Ada for the Second TDRSS Ground Terminal project at Goddard. A representative from Logicon related that there was much emphasis on risk management in the study of Ada by the FAA. The FAA also ultimately decided to use Ada for their Advanced Automation System.

Another transition issue is the "conversion" of programmers to Ada. Programmers are known to often be quite loyal to a particular language. However, Frank McGarry noted that once people begin to use Ada on real projects, they do not want to go back to the language they used before. Ed Seidewitz mentioned that Rational had begun early development with a large number of LISP programmers, who became strong Ada converts and refused to maintain their previous LISP code.

There can be, nevertheless, considerable resistance to the switch to Ada. A representative from PRC commented that experienced C and Pascal programmers consider Ada to have "too much overhead" and they complained that "Ada was designed to control the programmer." Gary Walker remarked that the transition from MODULA II to Ada is easier. MODULA is now taught in several schools.

Methodology

There is an increasing emphasis on the use of object-oriented design with Ada. However, there was some concern in the audience about the maturity of object-oriented methodologies.

Ed Seidewitz replied that the problem is partly that different people mean different things by the term "object-oriented design." Nevertheless, there are some important, useful concepts which are common to all

object-oriented approaches, such as abstraction and encapsulation. The object-oriented methodology developed by and used in the Flight Dynamics Division at Goddard has proven effective so far, though more experience is needed on judging the quality of proposed designs.

Kathy Rogers commented that a major issue is the scaling up of object-oriented approaches to larger and more complex systems. Eric Booth of CSC stated that they had run into a wall with the original object-oriented approaches at sizes of 200 to 300 thousand lines of code. However, much of this problem could be overcome by the use of the object-oriented "subsystem" concepts. Ed Seidewitz indicated that with such techniques, he believes object-oriented design can readily scale up to large systems.

Training

Several speakers during the symposium stressed the importance of effective training and especially the gaining of hands-on experience in the use of Ada. The panelists were asked how big they felt a training project had to be to give new Ada programmers practical experience.

Frank McGarry felt that the Electronic Message System (EMS) project used for early training in the Goddard Flight Dynamics Division was of marginal size at 8 to 10 thousand lines of code. Ed Seidewitz remarked that EMS would have been a better exercise if it had been more directly applicable to the application domain of the division. However, such training projects are often difficult to formulate.

Glenn Freedman commented from the audience that the real scaling issue was complexity, not size. He believes that a good pilot project is a complete Ada Artifact, such as that being considered by the Software Engineering Institute, on which students can build.

Reuse

There was a strong interest in ways to promote the reuse of code across projects. However, there was also a feeling that current contracting approaches discourage this. Bob Nelson expressed the need for contractual mandates for reuse.

Effective reuse also requires a common repository of quality reusable components. Cora Carmody from PRC mentioned that the space station Software Support Environment (SSE) will apply qualification criteria to software in its reuse library. Components will have to meet both functionality and complexity requirements. The exact method for doing this is still under development.

Kathy Rogers commented that the space station project also plans to reuse more than code. This includes the reuse of such things as requirements and staffing plans.

Real-Time

There was considerable discussion of the use of Ada in embedded, real-time applications. There are still concerns with the performance of Ada in time critical situations, especially when tasking is involved. The panel seemed to feel that the problems right now were mostly with poor implementations, rather than with flaws in the language itself.

Frank McGarry stated that he felt that Ada implementations were not yet ready for real-time applications, but that most software does not have real-time requirements. On the other hand, Bob Nelson said that these issues were being addressed for the space station through ongoing prototyping, and that early indications are that Ada is OK for real-time.

Dan Roy of Ford Aerospace commented from the audience on the great improvement certain implementations have made in reducing the time for a synchronous rendezvous, down to 25-500 microseconds. He also

mentioned that if one has problems with tasking, it is possible to do real-time applications using a non-tasking subset of Ada. This should be just as easy as doing these applications in other non-tasking languages, with similar performance.

Stephen Leake from the National Institute for Standards and Technology described his work on the use of Ada for NASA Flight Telerobotic Servicer robotics software. At Goddard they are currently reimplementing a robotic control system in Ada. He believes that the Ada system is much better than the original and that the execution speed is good.

There was general agreement that it is very important to choose a good compiler if you need to make effective use of tasking. However, there was still some concern with the fundamental Ada tasking paradigm for hard real-time applications. There was disagreement on how far the Ada 9X standard revision will go in altering the tasking model, though the Ada 9X process will certainly address tasking issues.

Besides execution speed, there were some remarks on the varying Ada source-to-machine-instruction expansion ratios presented by various speakers. Kathy Rogers commented that this is highly implementation dependent and that it is improving. However, Dan Roy responded that he did not feel that such expansion ratios were really important measures, and Bill Howle did not even consider them valid.

Conclusion

To conclude the session, the moderator asked each panelist how he or she would advise a new NASA administrator to ease the transition to Ada.

Gary Walker felt that NASA headquarters should not make edicts, but should give support to the centers.

Michael Holloway throught that it was important for Langley to catch up to the other centers in the use of Ada.

Bill Howle stated that the most important thing is to promote education and training, to both technical and management personnel.

Frank McGarry felt that NASA headquarters should go beyond just supporting the use of Ada, and actually mandate Ada as the common NASA language.

Bob Nelson, however, was uncomfortable with the idea of a mandate, saying that people in NASA are not used to such dictates from headquarters. He stressed, instead, the importance of incentives to promote the use of Ada.

Finally, Kathy Rogers felt that NASA should revisit the software development life cycle and replace the inadequate waterfall model.

Appendix B:

ATTENDEES OF THE FIRST NASA ADA USERS' SYMPOSIUM

ATTENDEES

Alanen, Jack	Sohar, Inc.
Amsler, John	OAO Corp.
Anderson, Marshall	Dept. of Defense
Badal, David	Lockheed Missiles & Space Co.
Barber, Gary	Intermetrics, Inc.
Barksdale, Joseph	NASA/GSFC
Bartlett, Tom	GSFC/NASA
Bates, Eileen	IDE, Inc.
Beall, Daniel	Ford Aerospace Corp.
Bennett, Toby	Ford Aerospace Corp.
Blue, Velma	Defense Communications Agency
Bobzien, Gale	PSC
Bognar, Jeff	DCA/JDSSC/C344
Booth, Eric	Computer Sciences Corp.
Bradley, Stephen	MMS Systems
Brady, Talbot	Jet Propulsion Lab
Brechbiel, Fred	Computer Sciences Corp.
Bredeson, Mimi	Space Telescope Science Inst.
Bredeson, Richard	Omitron
Brierschmitt, Michael	Ford Aerospace Corp.
Brinker, Elisabeth	NASA/GSFC
Britt, Chester	Defense Communications Agency
Brophy, Carolyn	University of Maryland
Brown, David	Auburn University
Brown, James	Jet Propulsion Lab
Burt, Roger	Jet Propulsion Lab
Butler, Madeline	NASA/GSFC
Carmody, Cora	Planning Research Corp.
Carr, Maureen	McDonnell Douglas Astronautics Co.
Carroll, Rossye	Computer Sciences Corp.
Caughel, Brian	Cadre Technologies
Cernosek, Gary	McDonnell Douglas Astronautics Co.
Chang, Joan	Computer Sciences Corp.
Chen, Jennifer	Computer Sciences Corp.
Chiang, Ted	Ford Aerospace Corp.
Chu, Richard	Ford Aerospace Corp.
Church, Vic	Computer Sciences Corp.
Cisney, Lee	NASA/GSFC
Clark, David	Unisys Corp.
Clema, Joe	IITRI
Colaizzi, Donald	Computer Sciences Corp.
Court, Terry	Hughes Aircraft Company
Cross, James	Auburn University
Cuddie, Jim	Martin Marietta
Cupak, John	HRB Systems

Daniell, Walter	IBM
Daniels, Catherine	Defense Communications Agency
Diclaudio, Mary	Jet Propulsion Lab
Drew, Dan	Unisys Corp.
Driesman, Debbie	Computer Sciences Corp.
Dyer, Kevin	Adanet
Ebker, Keith	Computer Sciences Corp.
Edelstein, E.	Grumman Data Systems
Edgar, Eric	HRB Systems
Ellis, Walter	IBM
Emerson, Curtis	NASA/GSFC
Emmart, Connie	Computer Sciences Corp.
Esker, Linda	Computer Sciences Corp.
Evers, Jay	Unisys Corp.
Ferguson, Frances	Stanford Telecommunications Corp.
Fermino, Kerri	Stanford Telecommunications Corp.
Ferry, Dan	Computer Sciences Corp.
Finnegan, Kenneth	Martin Marietta
Firsching, Dorothy	PRC
Fly, Ken	NASA/GSFC
Formanek, Kathleen	Martin Marietta
Freedman, Glenn	University of Houston at Clear Lake
Gacuk, Peter	Spar Aerospace
Garcia, Enrique	Jet Propulsion Lab
Gardner, Michael	Computer Sciences Corp.
Gilliland, Denise	Stanford Telecommunications Corp.
Gilyeat, Colin	Advanced Technology, Inc.
Girone, Chuck	GE Astro Space
Godfrey, Sally	NASA/GSFC
Goldberg, Nancy	Computer Sciences Corp.
Gordon, Marc	Booz, Allen & Hamilton
Grafton, Ed	Link Flight Simulation Corp.
Graves, Rusell	Dept. of Defense
Griswold, Robert	Computer Technology Associates
Guenterberg, Sharon	Planning Research Corp.
Gupta, Lakshmi	Ford Aerospace Corp.
Hain, Gertrud	
Hain, Klaus	
Hall, David	Computer Based Systems, Inc.
Hall, Gardiner	Ford Aerospace Corp.
Halterman, Karen	OAO Corporation
Haney, Modenna	Martin Marietta
Harris, Bernard	NASA/GSFC
Hartman, Ken	Computer Sciences Corp.
Hebenstreit, Karl	Logican, Inc.
Heffernan, Henry	GCN
Heyliger, George	Computer Technology Associates
Higgins, Herman	Dept. of Defense

Holloway, Michael	NASA/LaRC
Holmes, James	Unisys Corp.
Howle, Bill	NASA/MSFC
Huber, Hartmut	NSWC
Hutchison, Roberta	The Mitre Corp.
Iseman, Chelsea	Defense Communications Agency
Jackson, Laverne	PRC
Janaczek, Mark	Martin Marietta
Jaworski, Allan	Software Productivity Consortium
Jessen, William	RCA - ESD
Johannson, Hank	Ford Aerospace Co.
Kannappan, Sam	ABI Enterprises
Kathuria, Manbir	
Kelly, John	Jet Propulsion Lab
Kelly, Lisa	NASA/GSFC
Kelly, Nancy	PSC
Kim, Seung	Computer Sciences Corp.
Kirby, Philip	NASA/GSFC
Kirk, Daniel	NASA/GSFC
Klein, Camille	Hughes Aircraft Co.
Klitsch, Gerald	Computer Sciences Corp.
Kubaryk, Peter	IITRI
Kudlinski, Robert	NASA/LaRC
Labaugh, Robert	Martin Marietta Aerospace Corp.
Lavallee, David	Ford Aerospace & Comm. Corp.
Leake, Stephen	NIST
Ledford, Rick	McDonnell Douglas Corp.
Lee, Sophia	Defense Communications Agency
Lee, Tom	NASA/GSFC
Leenhouts, Kathleen	General Electric
Liebhardt, Edward	
Lin, Chi	Jet Propulsion Lab
Lin, Meng-Chun	Integral Systems, Inc.
Littman, Dave	NASA/GSFC
Liu, Kuen-San	Computer Sciences Corp.
Lloyd, Michael	General Dynamics
Loesh, Bob	System Technology Institute
Lowe, Dawn	NASA/GSFC
Mall, Vance	Independent Consultant
Mallet, Bob	Technology Planning, Inc.
Mangieri, Mark	Johnson Space Center
Marciniak, John	Marciniak & Associates
Martinez, Bill	Ford Aerospace Corp.
Mathiasen, Candy	Unisys Corp.
Maury, Jesse	NASA/GSFC
McComas, Dave	NASA/GSFC
McCullough, Sterling	Computer Technology Group
McDonald, Beth	Dept. of Defense

McGarry, Frank	NASA/GSFC
McKeag, Thomas	HRB Systems
Mixon, Don	The Mitre Corp.
Mohrman, Carl	Martin Marietta ATC
Molko, Patricia	Jet Propulsion Lab
Montoya, Maria	McDonnell Douglas Astronautics Co.
Moore, Mike	CTA, Inc.
Mularz, Diane	The Mitre Corp.
Murphy, Robert	NASA/GSFC
Naab, Joseph	NASA/GSFC
Nelson, Robert	NASA Space Station Program Office
O'Brien, David	Concurrent Computer Corp.
Osman, Jeffrey	Jet Propulsion Lab
Owens, Kevin	PRC
Owings, Jan	NASA/GSFC
Patel, Kant	Computer Sciences Corp.
Peters, Karl	NASA/GSFC
Pincosy, John	Data Systems Analysis
Pixton, Jerry	Unisys Corporation
Plunkett, Theresa	Dept. of Defense
Puleo, Joe	Concurrent Computer Corp.
Ransom, Bert	NASA/GSFC
Rennie, Tom	NASA/GSFC
Reph, Mary	NASA/GSFC
Rice, Raymond	McDonnell Douglas Astronautics Co.
Rigney, Brandon	PRC
Ritter, Sheila	NASA/GSFC
Roberts, Becky	PRC
Robertson, Laurie	Computer Sciences Corp.
Robinson, Mary	The Mitre Corp.
Robison III, W.	Jet Propulsion Lab
Rogers, Kathy	The Mitre Corp.
Rohr, John	Jet Propulsion Lab
Roy, Daniel	Ford Aerospace Corp.
Rucki, Dan	Dept. of Defense
Sabnis, Releha	Computer Sciences Corp.
Sank, Victor	FHA
Schubert, Kathy	NASA/LeRC
Schwenk, Robert	NASA/GSFC
Seeger, Howard	Science Applications International Corp.
Seidewitz, Ed	NASA/GSFC
Seo, Kyungsil	Defense Communications Agency
Severino, Tony	General Electric/RCA
Shen, Vincent	MCC
Skinner, Judith	Jet Propulsion Lab
Smalling, Richard	
Smith, David	Hughes
Snyder, Glenn	OAO Corporation

Soloman, Carl	NASA/GSFC
Spence, Bailey	Computer Sciences Corp.
Stammerjohn, Amy	Grumman/PCS
Stammerjohn, L.	The Mitre Corp.
Stanley, Carolyn	Martin Marietta
Stark, Michael	NASA/GSFC
Steinbacher, Jody	Jet Propulsion Lab
Stevenson, Jeff	Martin Marietta
Stickle, Richard	HEI
Szulewski, Paul	C. S. Draper Labs, Inc.
Tasaki, Keiji	NASA/GSFC
Thackery, Kent	Planning Analysis Corp.
Thompson, John	Ford Aerospace Corp.
Tindal, M.	NASA/GSFC
Trocki, Martin	Intermetrics
Tsounos, Andrew	SEI
Tupper, Burr	Intermetrics
Usavage, Paul	General Electric
Venkataraman, Ravi	ST Systems Corp.
Vernacchio, Al	NASA/GSFC
Vladavsky, Luba	Logicon, Inc.
Wackley, Joseph	Jet Propulsion Lab.
Walden, G.	Aerospace Corp.
Waligora, Sharon	Computer Sciences Corp.
Walker, Harry	Jet Propulsion Lab
Walker, Scott	IDE, Inc.
Wall, Doug	IDE, Inc.
Wallace, Charles	Raytheon Service Co.
Weisman, David	Unisys Corp.
Welborn, Richard	Stanford Telecommunications, Inc.
Wilson, Jean	MDAC/KSC
Wong, Yuen Yi	Defense Communications Agency
Wood, Richard	Computer Sciences Corp.
Yang, Chao	NASA/GSFC
Young, Eugene	NASA/GSFC
Zahn, Maryanne	HEI

Appendix C:

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

[illegible]

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-004, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-78-302, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3), W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

PRECEDING PAGE BLANK NOT FILMED

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1), W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-706, Annotated Bibliography of Software Engineering Laboratory Literature, G. Heller, January 1989

SEL-83-001, An Approach to Software Cost Estimation, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, Proceedings From the Ninth Annual Software Engineering Workshop, November 1984

SEL-85-001, A Comparison of Software Verification Techniques, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, R. Murphy and M. Stark, October 1985

SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985

SEL-85-004, Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, R. W. Selby, Jr., May 1985

SEL-85-005, Software Verification and Testing, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, Proceedings From the Tenth Annual Software Engineering Workshop, December 1985

SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

SEL-86-002, General Object-Oriented Software Development, E. Seidewitz and M. Stark, August 1986

SEL-86-003, Flight Dynamics System Software Development Environment Tutorial, J. Buell and P. Myers, July 1986

SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986

SEL-86-005, Measuring Software Design, D. N. Card, October 1986

SEL-86-006, Proceedings From the Eleventh Annual Software Engineering Workshop, December 1986

SEL-87-001, Product Assurance Policies and Procedures for Flight Dynamics Software Development, S. Perry et al., March 1987

SEL-87-002, Ada Style Guide (Version 1.1), E. Seidewitz et al., May 1987

SEL-87-003, Guidelines for Applying the Composite Specification Model (CSM), W. W. Agresti, June 1987

SEL-87-004, Assessing the Ada Design Process and Its Implications: A Case Study, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-008, Data Collection Procedures for the Rehosted SEL Database, G. Heller, October 1987

SEL-87-009, Collected Software Engineering Papers: Volume V, S. DeLong, November 1987

SEL-87-010, Proceedings From the Twelfth Annual Software Engineering Workshop, December 1987

SEL-88-001, System Testing of a Production Ada Project: The GRODY Study, J. Seigle and Y. Shi, November 1988

SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988

SEL-88-003, Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis, K. Quimby and L. Esker, December 1988

SEL-RELATED LITERATURE

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," Proceedings of the First Pan-Pacific Computer Conference, September 1985

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," Journal of Systems and Software, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," Proceedings of the International Computer Software and Applications Conference, October 1985

[4]Basili, V. R., and D. Patnaik, A Study on Fault Prediction and Reliability Assessment in the SEL Environment, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," Communications of the ACM, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and J. Ramsey, Structural Coverage of Functional Testing, University of Maryland, Technical Report TR-1442, September 1984

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P--A Prototype Expert System for Software Engineering Management," Proceedings of the IEEE/MITRE Expert Systems in Government Symposium, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press, 1979

[5]Basili, V. and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," Proceedings of the 9th International Conference on Software Engineering, March 1987

[5]Basili, V. and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," Proceedings of the Joint Ada Conference, March 1987

[5]Basili, V. and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, June 1988

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., Comparing the Effectiveness of Software Testing Strategies, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R. and R. W. Selby "Four Applications of a Software Data Collection and Analysis Methodology," Proceedings of the NATO Advanced Study Institute, August 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, July 1986

[5]Basili, V. and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, December 1987

[2]Basili, V. R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: IEEE Computer Society Press, 1978

[5]Brophy, C., W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the Joint Ada Conference, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," Proceedings of the Washington Ada Technical Conference, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, October 1985

[5]Card, D. and W. Agresti, "Resolving the Software Science Anomaly," The Journal of Systems and Software, 1987

[6]Card, D. N., and W. Agresti, "Measuring Software Design Complexity," The Journal of Systems and Software, June 1988

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," IEEE Transactions on Software Engineering, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," ACM Software Engineering Notes, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: IEEE Computer Society Press, 1983

[5]Doubleday, D., "ASAP: An Ada Static Source Code Analyzer Program," University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S. and C. Brophy, "Experiences in the Implementation of a Large Ada Project," Proceedings of the 1988 Washington Ada Symposium, June 1988

Hamilton, M., and S. Zeldin, A Demonstration of AXES for NAVPAK, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

Jeffery, D. R., and V. Basili, "Characterizing Resource Data: A Model for Logical Association of Software Data," University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," Proceedings of the Tenth International Conference on Software Engineering, April 1988

[5]Mark, L. and H. D. Rombach, "A Meta Information Base for Software Engineering," University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L. and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[5]McGarry, F. and W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," Proceedings of the Hawaiian International Conference on System Sciences, January 1985

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (Proceedings), March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," Proceedings of the Eighth International Computer Software and Applications Conference, November 1984

[5]Ramsey, C. and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," IEEE Transactions on Software Engineering, March 1987

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," Proceedings From the Conference on Software Maintenance, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," Proceedings of the 21st Hawaii International Conference on System Sciences, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," Proceedings of the CASE Technology Conference, April 1988

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications, October 1987

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," Proceedings of the Joint Ada Conference, March 1987

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

[5]Valett, J. and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," IEEE Transactions on Software Engineering, February 1985

[5]Wu, L., V. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," Proceedings of the Joint Ada Conference, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," Journal of Systems and Software, 1988

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

NOTES:

[1]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985.

[4]This article also appears in SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986.

[5]This article also appears in SEL-87-009, Collected Software Engineering Papers: Volume V, November 1987.

[6]This article also appears in SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988.